DALENCOURT Alex
GOLDAK Ludovic

TP2 Recherche de plus court chemin

Les deux algorithmes de Prim et Kruskal ont été implémenté.

Conception des algorithmes

Prim

L'algorithme de prim est implémenté dans la classe java Prim du package algorithme.

Cet algorithme stock dans une liste tous les nœud actuellement rencontré et dans une seconde les liens qui ont été sélectionnés. A chaque étape une stack est mise à jour avec les liens disponibles à partir graphe actuellement réalisé. Enfin, lorsqu'un nœud est ajouté au graphe on supprime les cycles possibles de la stack des liens disponibles.

```java
@Override
public Graph startAlgorithm(final Graph graph) throws VertexNotFoundException {
    List<Vertex> vertexInsere = new ArrayList<>();
    List<Edge> edgeInsere = new ArrayList<>();

    Vertex current = graph.getVertexList().remove( i: 0);
    vertexInsere.add(current);
    List<Edge> liensDisponibles = graph.getEdgesFromVertex(current);
    liensDisponibles.sort(edgeComparator);

    while(graph.getCountVertex() > 0) {
        Edge tmp = Collections.min(liensDisponibles, edgeComparator);
        Vertex target = vertexInsere.contains(tmp.getSource()) ? tmp.getTarget() : tmp.getSource();
        graph.getVertexList().remove(target);
        vertexInsere.add(target);
        edgeInsere.add(tmp);
        liensDisponibles.addAll(graph.getEdgesFromVertex(target));
        supprimerCycles(vertexInsere, liensDisponibles);
    }

    final Graph output = new GraphImpl();
    output.getVertexList().addAll(vertexInsere);
    output.getLinks().addAll(edgeInsere);
    return output;
}
```

Cette méthode est peu gourmande en espace mémoire car elle nécessite la création de très peu d'objet qui se limite à la création de quelques conteneurs toutefois elle nécessite réorganiser à chaque itération le liens en fonction du poid pour toujours prendre le plus faible disponible ce qui fait perdre en temps.

Kruskal

L'algorithme de Kruskal est implémenté dans la classe java Kruskal du package algorithme.

Cet algorithme parcours l'ensemble des liens orientés par poid croissant. Un élément VertexGroup

permet de regrouper en sous graphe les différents nœuds parcourus selon la méthode de Kruskal.

Afin de limiter la création d'objets en cas fusion de deux groupes le contenu d'un groupe est recopier dans l'autre réduisant ainsi l'utilisation de mémoire.

```java
@Override
public Graph startAlgorithm(final Graph graph) {
    final List<VertexGroup> fusions = new ArrayList<>();
    for (Iterator<Edge> it = graph.getSortedEdgeIterator(); it.hasNext(); ) {
        if(fusions.size() == 1 && fusions.get(0).getVertexs().size() == graph.getCountVertex()){
            break;
        }
        final Edge edge = it.next();
        VertexGroup src = null;
        VertexGroup tgt = null;
        for(VertexGroup group : fusions){
            if(group.getVertexs().contains(edge.getSource())){
                src = group;
            }
            if(group.getVertexs().contains(edge.getTarget())){
                tgt = group;
            }
        }
        if(src == null && tgt == null){
            fusions.add(new VertexGroup(edge));
        } else if (src == tgt){
            continue;
        } else if (src == null){
            tgt.fusion(edge.getSource(),edge);
        } else if (tgt == null){
            src.fusion(edge.getTarget(),edge);
        } else {
            src.fusion(tgt,edge);
            fusions.remove(tgt);
        }
    }
```

Ainsi cet algorithme ne nécessite pas de réorganiser les liens en fonction du poids de ces dernier. Et gagne donc en temps d'exécution mais perd en utilisation de mémoire car nécessite un nombre de création de nœud temporaire qui est de au maximum de n / 2 avec n le nombre de nœuds.

Tests de performance

Le jar exécutable TestPerformance.jar permet d'exécuter les algorihtme :

java -jar {chemin vers TestPerfomance.jar} {algorithme, lettre: P (pour Prim), K (pour Kruskal)} {nombre de graphes à calculer} {nombre de nœuds} {probabilité d'une arrête}

Pour contrôler la validité des algorithmes le programme affiche le graphe d'origine et graphe de sortie, il reste plus qu'à l'utilisateur de vérifier que le graphe en sortie est bien le graphe recouvrant de poids le plus faible. Il est possible que le générateur aléatoire de graphe génère des graphes non connexes. Si c'est le cas suivant l'algorithme les parties non connecté au graphe peuvent être omises en fonction de l'algorithme.

DALENCOURT Alex
GOLDAK Ludovic

Le programme affiche le temps d'exécution du calcul pour chaque graphe ainsi que le temps moyen total d'exécution des algorithmes. Pour des graphes de grande taille il n'est pas étonnant que le programme prenne du temps à s'exécute mais que le temps relevé pour l'exécution de l'algorithme soit court. En effet, la génération de graphe aléatoire de grande taille dépend de la taille de n^2.

Tests d'espace mémoire :

Comme expliqué dans l'implémentation des algorithmes, Kruskal nécessite d'avantage d'espace mémoire que pour Prim.

Test de temps d'exécution :

Après plusieurs tests nous nous sommes aperçu que le temps de réalisation de plusieurs analyses à la suite pour une exécution du programme diminuait afin tendre vers 0 ms. Cela est probablement provoqué par la gestion de la mémoire java. Afin de limiter les effets de ce comportement nous forçons le nettoyage de garbage collector entre chaque calcul.

Les tests seront effectués selon les variables suivantes :

- Nombre d'itérations : 10

- Nombre de nœuds : 100 / 200 / 400 / 500

- Probabilité de liens : 0,1 / 0,3 / 0,5 / 0,7 / 0,9

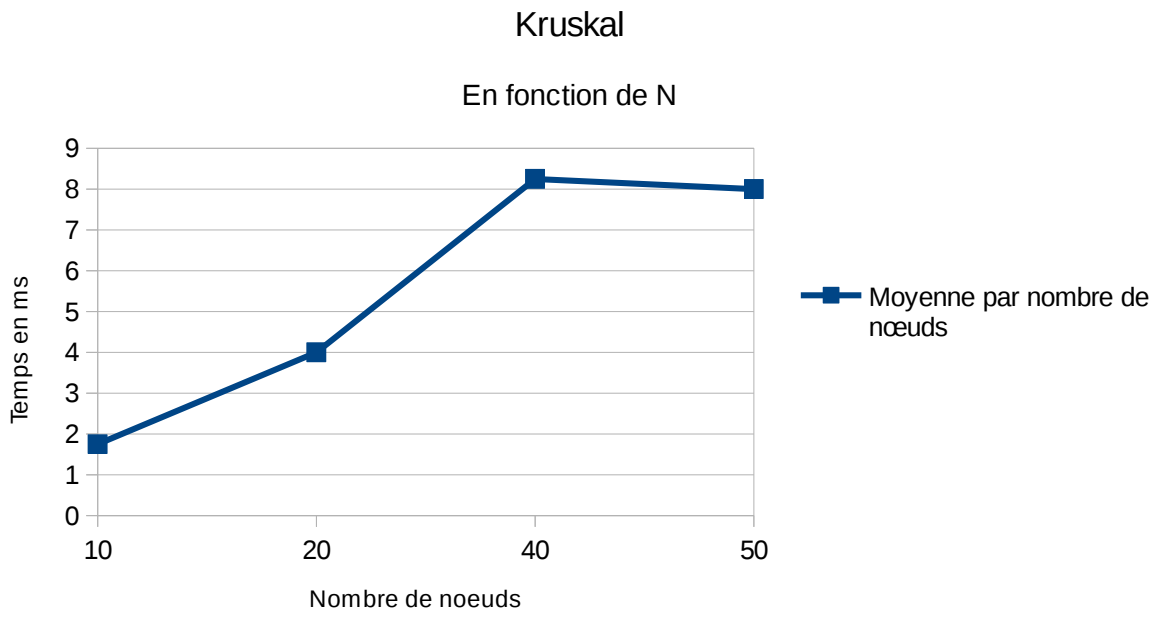
le script script.sh permet d'exécuter la suite des tests de perfomances :

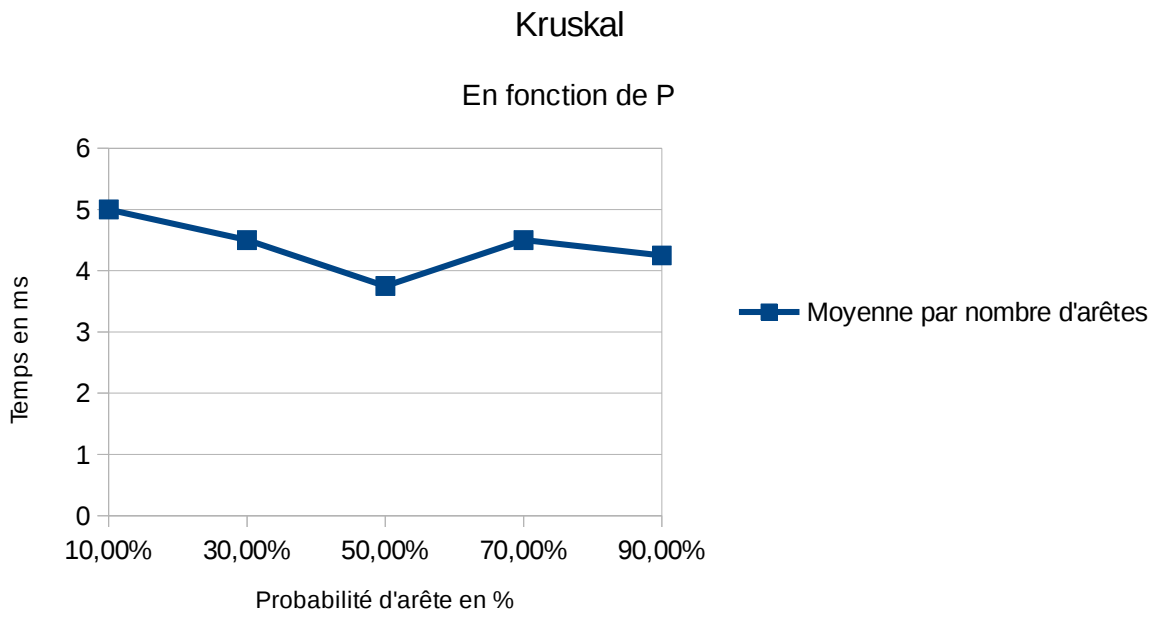
./script {chemin vers le jar} {algorithme à utiliser}

Les fichiers performanceKruskal.plot et performancePrim.plot contient les temps d'exécution pour chaque paramètres.

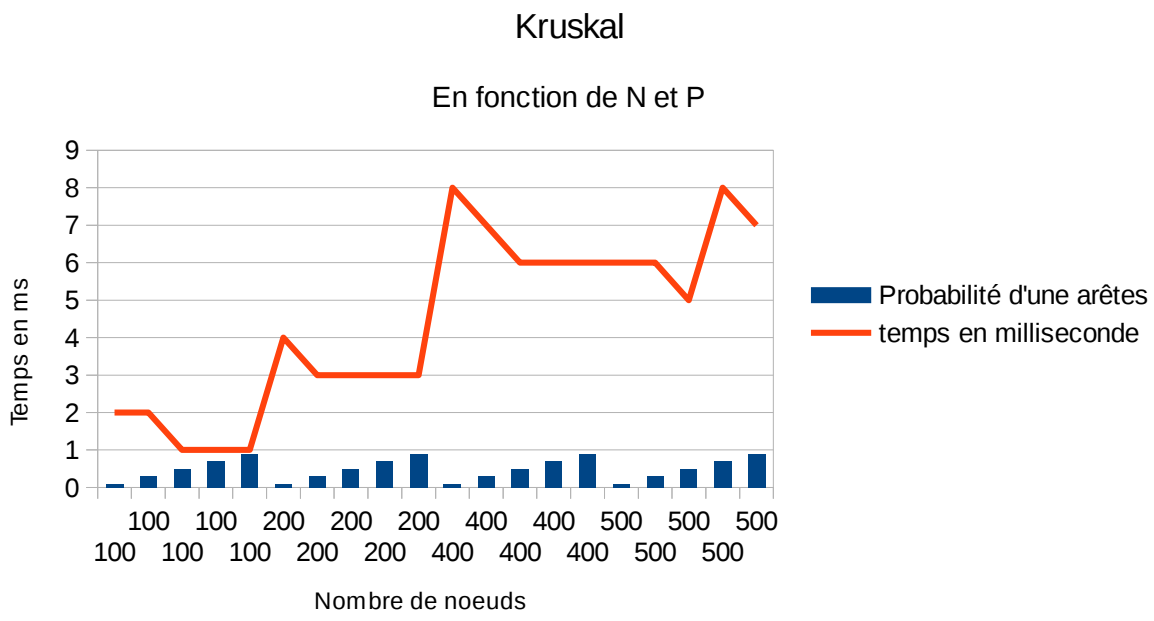
Voici des graphiques pour se rendre compte de l'évolution en temps de chaque algorithmes en fonction du nombre de nœuds et de la probabilité des arrêtes.

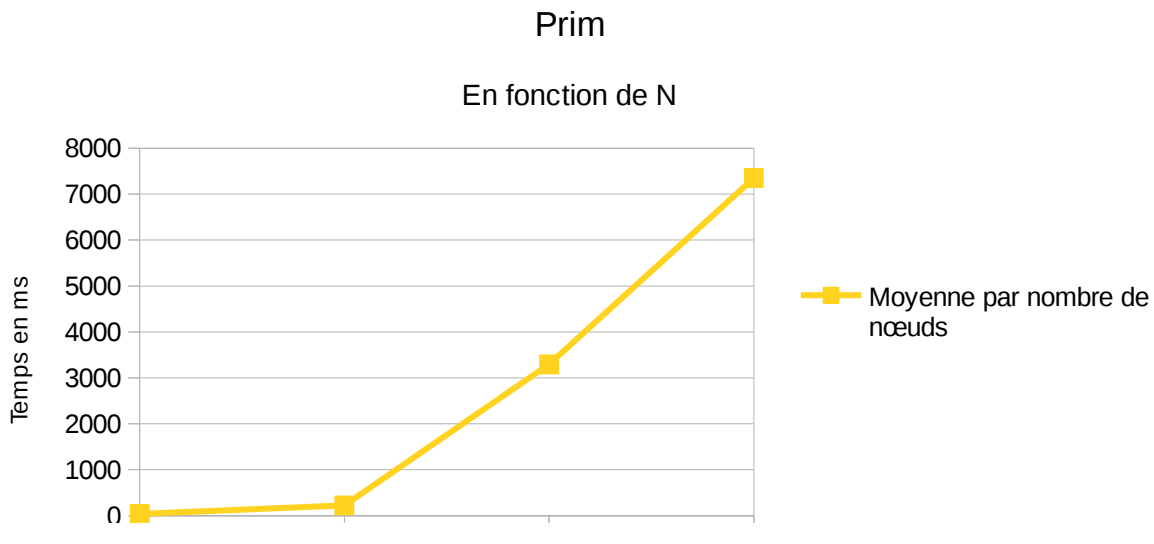
DALENCOURT Alex
GOLDAK Ludovic

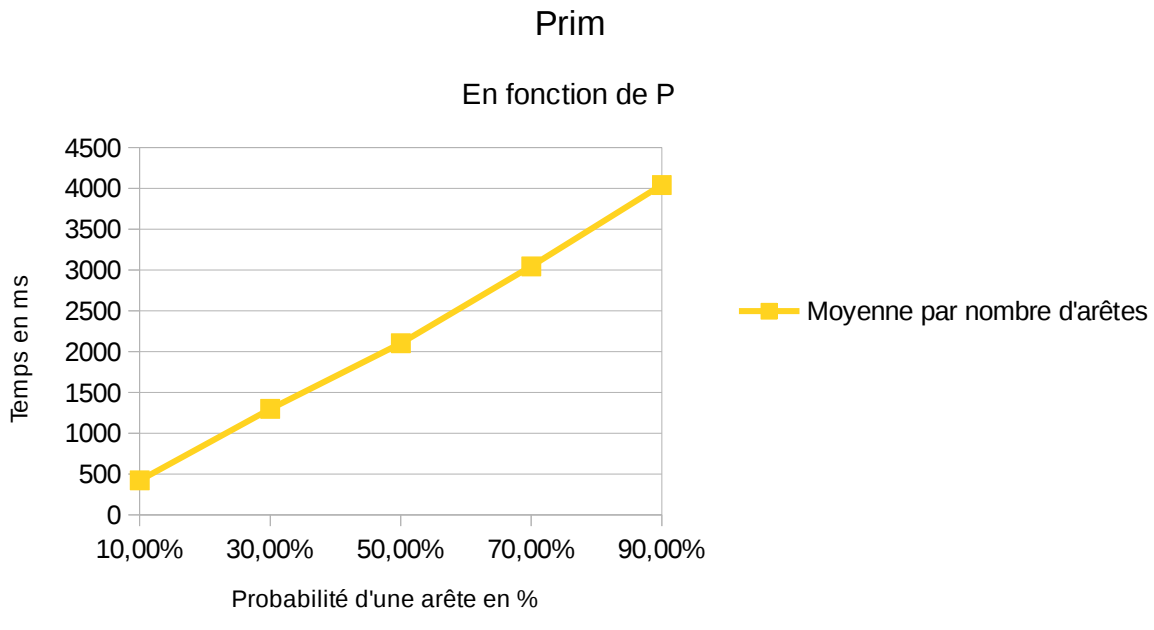




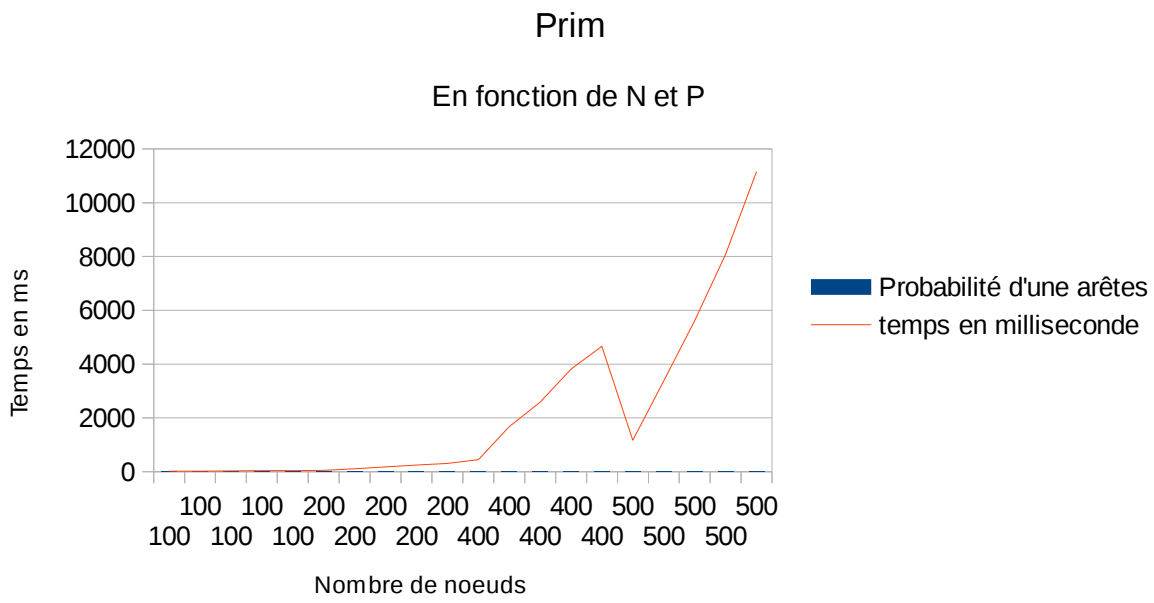
DALENCOURT Alex
GOLDAK Ludovic







DALENCOURT Alex
GOLDAK Ludovic



A la vu de la courbe des courbes de temps l'algorithme de Kruskal est en tout point plus performant que l'algorithme de Prim. En effet, ce dernier évolue en tant constant peu importe le nombre d'arêtes (Le programme s'arrête dès que tous les nœuds ont été fusionnés). De plus même si le temps dépend du nombre de nœuds ce facteur ne semble pas avoir une importante influence sur le programme.

Prim est nettement influencé par le nombre de nœuds et encore plus par le nombre d'arêtes.