RAPPORT ALGORITHME TEST DE PLANARITE

Par Remi Herve, Ludovic Marquet et Valentin Jubert

I. Analyses

En espace, l'algorithme stocke le graphe sous forme de liste d'adjacence. Nous stockons donc les sommets, et les voisins sont représentés par des pointeurs sur les sommets voisins. Malheureusement, par soucis de simplicité algorithmique, un doublon des sommets est créé pour le second graphe.

En temps, l'algorithme parcourt à plusieurs reprises les sommets ou les arêtes d'une partie du graphe initial. Le calcul de composantes connexes s'exécute en O(n+m) (en cas de multiples composantes, le parcours ne visite le graphe qu'une seule fois). L'instruction plonger s'exécute en O(n+m) (le temps pour trouver un chemin), mais on ne l'exécute que pour un seul fragment lors de la boucle while. Le calcul des faces admissibles parcourt toutes les faces, donc il s'exécute en O(n+m). Ce calcul est exécuté pour chacun des fragments. Puisque le nombre de fragments dépend de n et de m, on peut dire que le calcul des faces admissibles pour chaque face s'exécute en $O((n+m)^2)$. La boucle while extérieure dépendant du nombre de fragments, on peut dire que la boucle while s'exécute en $O((n+m)^3)$, et c'est la complexité de l'algorithme. A l'extérieur de cette boucle, l'initialisation des faces se faisant en O(n), il y a la lecture du fichier : O(n+m), et l'affichage : O(1).

II. Problèmes et solutions

Nous avons rencontré quelques problèmes lorsqu'il a fallu faire des calculs avec le cycle. Nous avons essayé de faire un marquage, mais le code n'était plus assez clair. Nous avons essayé avec une liste, mais il y a eu des soucis lors de l'ajout d'un chemin, donc nous avons finalement décidé d'implémenter le cycle sous la forme d'un second graphe avec une copie des sommets.

Ensuite, un problème est survenu au niveau du calcul des fragments. Nous avons créé un graphe représentant la différence du graphe avec le cycle, puis on exécute un calcul de composantes connexes sur ce graphe.

Enfin, le dernier problème a été la mise à jour des faces à partir d'un chemin donné. En effet il a fallu créer deux faces à partir d'une seule et d'un chemin. Pour cela on parcourt la face à la manière d'un cycle et on divise l'ensemble des sommets de la face en deux parties, puis on ajoute le chemin dans les deux parties.

III. Explication méthodes complexes

Mise à jour Face

Entrée : Un chemin C, une face F (une liste de sommet)
Sortie : 2 Faces
Début
F1 <- Vide (Une liste sans doublons)
F2 <- Vide (Une liste sans doublon)
Si le dernier sommet du chemin est le premier à apparaitre dans la Face (par rapport au premier sommet de C)
    Inverser C
Pour chaque sommet v de la face {
    Si v est avant le premier sommet du chemin ou après le dernier sommet du chemin
        F1 <- F1 U V
    Si v est après le premier sommet du chemin et avant le dernier
        F2 <- F2 U V
    Si v est le premier sommet du chemin {
        Mettre v dans F2
        Mettre les sommets du chemin dans F1
    }
Si v est le dernier sommet du chemin
    Mettre v dans F2
}
Inverser C
Ajouter C \ {1er Sommet U Dernier sommet} dans F2
Retourner F1 et F2
Fin

IV. Exécution

Pour pouvoir exécuter le projet il est nécessaire d'avoir Java 8. Le programme demande un unique argument qui est l'adresse du fichier représentant le graphe. Pour compiler : javac *.java dans le dossier mainpackage. Ensuite pour exécuter, il faut se placer dans la répertoire parent, puis lancer : java mainpackage.Main fichier.graphe.