Christian Artin (1584854)

Gabrielle Bourdages (1579702)

INF4705 Analyse et conception d'algorithmes

Groupe: 2

**Travail Pratique 3**

Travail présenté à:

Simon Brockbank

École Polytechnique de Montréal

17 avril 2014

## Objectif du travail

Le 3e et dernier travail pratique consiste à résoudre un problème qui nous est présenté avec un algorithme de notre création. La mise en situation qui nous est présentée est en fait un problème avec des graphes non-orientés qu’on désire parcourir en entier en y plaçant des routeurs à des endroits stratégiques. Il faut optimiser le nombre de routeurs pour que chaque ordinateur (nœud dans le graphe) soit connecté au réseau. Ainsi, nous savons que le but de notre algorithme sera de parcourir le graphe de façon efficace et optimisée pour le bon placement de nos routeurs.

## Notre algorithme

Voici le pseudo-code de notre algorithme ainsi que l’analyse de complexité théorique des fonctions principales.

*// On débute par trouver le nœud ayant le plus de voisins, car il est plutôt évident qu’il s’agît d’un endroit stratégique pour placer un routeur*

**Pour chaque** noeud:

**si** le nombre de voisins de noeudCourrant **>** nombre de voisins de noeudDepart

noeudDepart **:=** noeudCourrant

*// La boucle précédente parcours tous les nœuds et effectue une vérification qui coûte O(1), donc on obtient donc une complexité théorique de O(n) pour trouver le nœud de départ*

ajouter noeudDepart à fileFEL

*// On effectue une fouille en largeur*

**tant que** fileFEL n’est pas vide

défiler le noeudCourant de fileFEL

doitPlacerRouteur := faux

pour tout voisin de noeudCourant

si voisin est blanc

doitPlacerRouteur := vrai

**si** noeudCourant doitPlacerRouteur

colorer noeudCourant en noir

incrémenter le compteurRouteur de noeudCourant

ajouter noeudCourant à pileColoré

**pour tout** nœud qui n’est pas déjà passé dans la file **dans** noeudCourant.Voisins

**si** noeud est blanc

colorer noeud en gris

incrémenter le compteurRouteur de noeud

ajouter noeud à pile

ajouter noeud à fileFEL

**sinon**

**pour tout** noeud **dans** noeudCourant.Voisins

ajouter noeud à fileFEL

*// La fouille en largeur doit passer par tous les nœuds du graphe et la complexité de l’action à effectuer à l’intérieur de la boucle dépend du nombre de voisin du nœud courant. En pire cas, on aurait que chaque nœud est connecté à tous les autres nœuds du graphe, et donc on aurait une complexité de O(n2), alors qu’en meilleur cas, chaque nœud n’aurait qu’un seul voisin et donc notre boucle aurait une complexité de O(n). Il n’y a pas de complexité supplémentaire, car chaque action effectué sur les nœuds voisins (par exemple : changer sa couleur) est élémentaire et donc en O(1).*

*//On dépile chaque nœud pour enlever les routeurs qui sont superflus*

**tant que** pileColoré n’est pas vide

dépiler le noeudCourant

peutEnleverRouteur := true

**si** noeudCourant est noir

**pour chaque** voisin de NoeudCourant

**si** le compteurRouteur est à 1

peutEnleverRouteur := faux

On sort de la boucle

**si** peutEnleverRouteur est vrai

colorer noeudCourant en gris

décrémenter compteurRouteur de noeudCourant

**pour chaque** voisin de noeudCourant

décrémenter compteurRouter de voisin

*// Encore une fois on passe par tous les nœuds en effectuant des actions élémentaires sur chaque voisins. Comme pour la boucle d’avant, nous avons donc une complexité en meilleur cas de O(n) et une complexité en pire cas de O(n2) pour la boucle.*

*// Le tout pour un total qui peut donc varier entre O(n2) et O(n).*

## Originalité de l’algorithme

Notre algorithme est plutôt simple, nous effectuons simplement une fouille en largeur en plaçant les routeurs rapidement sans trop se poser de questions et en empilant les nœuds au fur et à mesure dans un pile. Lorsque nous avons fini notre fouille en largeur, nous sommes certains que chaque ordinateur (nœud) est alimenté, mais nous savons que la solution n’est pas nécessairement optimale. Pour pallier à cela, nous dépilons simplement chaque nœud en vérifiant que, s’il s’agît d’un routeur, il est vraiment pertinent, sinon on peut le retirer de notre liste des solutions. Ainsi, nous avons un algorithme simple qui parcourt seulement trois fois tous nos nœuds et qui s’assure d’une solution qui est optimisée (par de nœuds superflus).