Projet LSINF1121 - Algorithmique et structures de données

-

Rapport final Mission 2

Groupe 3.2

Boris Dehem (5586-12-00)

Sundeep Dhillon (6401-11-00)

Alexandre Hauet (5336-08-00)

Jonathan Powell (6133-12-00)

Mathieu Rosar (4718-12-00)

Tanguy Vaessen (0810-14-00)

Année académique 2014-2015

Introduction

Dans le cadre du cours "Algorithmique et structures de données", il nous a été demandé d'implémenter un programme de dérivation formelle. Ce programme devra utiliser le type abstrait arbre pour représenter et manipuler les expressions analytiques.

1 Choix d'implémentation

Pour la représention de l'arbre binaire, nous avons décidé de réutiliser l'implémentation de RBinaryTree décrite dans DSAJ-5.

2 Diagramme de classes

Voir pdf Diagramme de classe.

3 Difficultés rencontrées

Grâce aux questions de la séance intermédiaire, nous avons décortiqué le problème ce qui nous a permis de préparer correctement le partie implémentation. Nous n'avons donc pas rencontré de réel problème lors de la partie développement de l'application.

4 Analyse de la complexité calculatoire

4.1 Complexité temporelle

Méthodes de la classe LinkedRBinaryTree

Les méthodes de la classe LinkedRBinaryTree sont de complexité $\Theta(1)$. Sauf les méthodes suivantes : size, search, toString. Soit n, le nombre de noeuds dans l'arbre. La complexité de ces méthodes est de $\Theta(n)$. Pour les méthodes size et toString, l'ensemble des noeuds de l'arbre seront obligatoirement visités. Pour la méthode search, dans le pire des cas (si l'algorithme ne trouve pas l'élément recherché), l'ensemble des noeuds seront parcourus.

Méthodes de la classe Derivator

La méthode derivate va parcourir l'ensemble des noeuds pour trouver une dérivée à calculer. Dans le pire des cas, l'algorithme ne va pas trouver de dérivée à calculer et donc parcourir l'ensemble des noeuds. On peut donc en déduire que la complexité temporelle dans le pire des cas est de $\Theta(n)$, n étant le nombre de noeuds dans l'arbre.

La méthode initDerivatedTree ne fait qu'initialiser un arbre binaire représentant une dérivée. Donc la complexité temporelle de la méthode est de $\Theta(1)$.

Méthodes de la classe AnalyticExpression (+ Number et Variable)

Les méthodes getValue, setValue, equals et toString définies dans la classe abstraite AnalyticExpression sont toutes de complexités $\Theta(1)$. Ensuite, les classes Number et Variable étendent la classe AnalyticExpression et implémentent chacune la méthode 'Derivate' dont la signature était fournie dans AnalyticExpression. La complexité de cette méthode est $\Theta(1)$.

Méthodes des classes "Operator"

Toutes les opérations de propagation de l'opérateur de dérivation D s'effectuent en un nombre d'opérations constant. Dès lors, les méthodes derivate des classes AddOperator, SubOperator, MulOperator, DivOperator, ExpOperator, CosOperator et SinOperator sont toutes de complexité $\Theta(1)$.

5 Répartition du travail

Rédaction du rapport : Alexandre et Mathieu Conception du programme : Boris, Jonathan, Sundeep et Tanguy (voir code pour détails)