M1 ALMA
Université de Nantes
2011-2012

Rapport de Projet : Programmation générative

Marguerite Alain, Pluchon Gwenael
Rincé Romain, Sebari Nabila

Université de Nantes
2 rue de la Houssinière, BP92208, F-44322 Nantes cedex 03, FRANCE

Table des matières

1 Choix de la structure concrète

Nous avons recherché la structure concrète la plus performante dans l'ensemble des opérations. Ne pouvant savoir quelle serait la nature de l'utilisation de cette classe, nous avons opté pour une structure offrant les meilleures performances pour l'implémentation du tas. Plusieurs solutions se présentaient pour une implémentation de tas binaire :

Tableau des pères Non approprié (dédié aux arborescence non-ordonnées).

Une liste d'adjacence Un noeud étant un élément (étiquette, père, liste des fils) de la liste. Ses perfomances sont dégradées lors d'opération comme la supression ($O(n)$ pour un tas binaire) ou l'accès à un noeud à partir de son étiquette ($O(n)$). Dans le cas d'une utilisation où les opérations principales aurait été l'ajout d'un fils par exemple, cette solution aurait pu être retenue.

Liste de fratrie Un noeud étant un élément (étiquette, père, liste des frères, premier fils) de la liste.

Tableau modulaire definition manquante Structure présentant les meilleurs complexités pour l'ensemble des opérations. Cette solution répond le mieux aux contraintes que nous avons exposés précedement. C'est donc elle que nous avons retenue.

1.1 Developpement du tas binaire

Une fois la sélection de la structure faite. Nous avons au cours des semaines adopté le plan de développement suivant :

1. Définition des prototypes des méthodes
2. Implémentation des méthodes

3. Rajout de l'itérateur
4. Rajout de l'allocateur et du comparator
5. Debug

Cette démarche nous a permis d'avancer rapidément au cours des première semaine. Les cours d'algorithmiques du 1er semestre encore en tête. Nous avons rapidement déployées les méthodes principales du tas binaire (Extraire, Tasser, Insérer...), sans trop nous soucier de la syntaxe C++ (nouvelle pour notre bînome). Nous avons amèrement regretté notre insoucience de ne jamais tester nos méthodes avant la 5ème ou 6ème semaine. Nous avons donc passé un temps important à débuguer un nombre de d'erreurs de syntaxe conséquent.

La création d'un itérateur a été un réel frein dans le développement du projet. Notre premier objectif était de construire un itérateur permettant de parcourir les valeurs dans un ordre trié dépendant de la classe `Compare`. Cependant un tel itérateur nécessitait de maintenir une structure de taille égale à celle du tas, ce qui s'avérait coûteux en mémoire. Nous avons donc décidé de nous orienter vers un itérateur `DFS` plus classique. Malgré tout nous avons eu du mal a maitriser la syntaxe C++ lié aux itérateurs et ils nous a fallu malgré tout un certain temps.

2 Généralisation des différentes structures d'arbres

Table des figures