

Compte rendu TP4 : TRI

Brought to you by Rémi GATTAZ & Aymeric VIAL-GRELIER

Introduction

Les objectifs n° 0,1,2, 3, 4, 5, 6 sont à réaliser dans cet ordre

Les objectifs facultatifs suivants peuvent être traités dans n'importe quel ordre.

- Objectif 7 : un autre tri, n'introduit rien de nouveau
- Objectif 8 : une généralisation
- Objectif 9 : diversification des modèles de "triable"
- Objectif 11 : une réflexion sur les coûts
- Objectif 10 : génération et analyse des coûts.

Les objectifs 9 et 11 sont les plus intéressants

Objectif 0 :

La classe Tri actuelle est capable de gérer n'importe quel type de donnée. Par contre, étant donnée que ceci est possible grâce à un polymorphisme avec la classe Object, il n'est pas possible d'assurer que les objets utilisés ou retournés par les fonctions sont compatibles ou équivalents.

Par exemple, rien ne garantit pour la fonction `protected abstract Object[] sort(Object[] t, Comparator c);` que l'Object comparator est capable de trier les objets dans le tableau T. Rien ne garantit non plus que le tableau retourné par la fonction sera du même type que celui passé en paramètre.

Nous avons donc rajouté de la généricité (horizontale) pour assurer ceci. Nous avons alors obtenu les méthodes suivantes dans la classe Trier.

```
protected abstract <T> T[] sort(T[] t, Comparator<T> c);  
public <T> T[] trier(T[] t, Comparator <T> c){ ... }  
protected static <T>int verifier(T[] t, Comparator<T> c){ ... }
```

Avec cette version, nous assurons que dans le cadre de chaque fonction, le type T est le même.

Dans l'interface triable, nous avons pu constater le même problème. En revanche, cette fois il fallait pouvoir assurer que les Objets retournés et utilisés par les fonctions étaient tous du même type. Nous avons donc rajouté une généricité (verticale) sur l'interface pour obtenir :

```
public interface Triable <T> {  
    public T newInstance();  
    public Comparator<T> comparator();  
    public String toString(T o);  
    public long count();  
    public void resetCount();  
}
```

Avec cette version, nous assurons que le type T utilisé par toutes les fonctions est le même.

Objectif 1 :

Nous avons créé la classe StringTriable. Cette classe implémente l'interface Triable. Il a donc fallu définir toutes les fonctions se trouvant dans l'interface dans la classe. C'est à dire :

- public String newInstance() : retourne une String comme étant un nombre aléatoire entre 1 et 10 000 000
- public Comparator comparator() : retourne le comparator standard de String (String.CASE_INSENSITIVE_ORDER)
- public String toString(String aString) : retourne la string passé en paramètre
- public long count() : non utilisé, cette fonction retourne simple 0.
- public void resetCount() : non utilisé, cette fonction ne fait rien

Ensuite, en utilisant le programme Trier et la classe StringTriable, nous avons pu alors obtenir des collections d'objets Triables.

File	Trieur
0	47796421
1	41146751
2	97221891
3	34326311
4	44332531
5	26968131
6	83172611
7	22459361
8	46317861
9	23912341
10	86356721
11	61990361
12	60337551
13	61403581
14	79312761
15	75061681
16	22705861
17	5507731
18	66400521
19	80248931

Objectif 2 :

Nous avons crée la classe MaximumSort. Nous l'avons ensuite utilisé pour trier des StringTrieable et
avons obtenu le résultat suivant :

File	Trieur
0	5739764
1	5084639
2	3015587
3	749670
4	2462523
5	9984512
6	7923392
7	216054
8	6212759
9	5190382
10	6325304
11	8572773
12	2793847
13	3063089
14	9403785
15	809718
16	5335157
17	8369697
18	37868
19	3330527

coût de : MaximumSort (StringTrieable)

elts=20

Comp=0

Affect=57

Time=279000ms - 20

7	216054
4	2462523
12	2793847
2	3015587
13	3063089
19	3330527
18	37868
1	5084639
9	5190382
16	5335157
0	5739764
8	6212759
10	6325304
3	749670
6	7923392
15	809718
17	8369697
11	8572773
14	9403785
5	9984512

Ce résultat est correct. En effet, la fonction de comparaison que nous utilisons proviens de String. Il s'agit donc d'un tri lexicographique.

Objectif 3 :

Avec une nouvelle version de StringTriable utilisant un Comparator qui trie par ordre croissant, on obtient le résultat suivant :

File		Trieur	
0	4419590	coût de : MaximumSort (StringTriable1)	
1	1858606	# elts=20	
2	6420584	Comp=209	
3	5014989	Affect=57	
4	8618172	Time=572000ms - 20	
5	6764409	9	1694542
6	8186544	16	1833354
7	1929102	1	1858606
8	3817589	7	1929102
9	1694542	14	2155253
10	4766400	19	2571339
11	7141079	17	2816591
12	4239104	8	3817589
13	5159290	12	4239104
14	2155253	0	4419590
15	5324099	10	4766400
16	1833354	3	5014989
17	2816591	13	5159290
18	7852074	15	5324099
19	2571339	2	6420584
		5	6764409
		11	7141079
		18	7852074
		6	8186544
		4	8618172

Pour réaliser le compteur, nous avons rajouté dans la classe StringTriable1\$Comparator un attribut compteur. A chaque fois que la méthode compare est appelé, ce compteur est incrémenté. Il est ensuite possible de récupérer sa valeur ou la remettre à 0 en utilisant les fonctions count et resetcount.

Pour pouvoir utiliser ce mécanisme de compteur, il a fallut également rajouter un attribut pComparator qui est un objet du type StringTriable1\$Comparator. Ceci permet de toujours retourner le même comparator lors de l'appel à la fonction comparator de cette classe. Les

méthodes count et resetcount de la classe utilisent alors les méthodes du même nom de pComparator.

Objectif 4 :

Dans cet algorithme, il est effectué $\frac{(n-1)*n}{2}$ tests de comparaison. Il a donc un complexité $O(n^2)$

Avec 20 éléments, on doit obtenir $\frac{(20-1)*20}{2} = 190$ tests de comparaison. C'est bien ce que l'on a obtenu.

Avec 40 éléments, on doit obtenir $\frac{(40-1)*40}{2} = 780$ tests de comparaison. C'est encore bien ce que l'on a obtenu.

Objectif 5 :

Les améliorations sur l'agorithme de Heapsort n'ont pas été faites.

Objectif 6 :
