Labo 07 Hanoï - Rapport

Sacha Bron - Valentin Minder // HEIG-VD // POO // 20.11.2014

Introduction

Le but de ce laboratoire est de créer un programme capable de résoudre le fameux problème des tours de Hanoï. De plus, les structures de données seront recrées à la main afin de mieux comprendre les méchanismes utilisés par la bibliothèque standard.

Méthodologie et implémentation

Il y a, comme pour beaucoup de problème en informatique, plusieurs manière d'aborder ce problème. Voici comment nous avons raisonné et procédé.

Structure de données

Comme les disques disques tours de Hanoï ne peuvent être mis ou enlevé que s'ils sont tout en haut, les piles (*stack*) semblent être la structure de données la plus favorable à ce problème.

Algorithme

Il existe plusieurs façons optimales de résoudre ce problème. Nous avons choisi une méthode récursive pour la facilité et la clarté de son implémentation.
Ainsi, le problème consistant à déplacer la tour d'une aiguille à l'autre peut se généraliser en un problème consistant à déplacer une pile de taille quelconque d'une aiguille à l'autre. Ainsi, le problème peut être décomposé. En effet, si nous devons déplacer une tour de n disques, cela revient à déplacer une tour de $n-1$ disques, de déplacer le plus gros disque, puis de replacer les disques plus petits dessus.

Classes principales

Notre programme est constitué de plusieurs classes ayant des buts précis.

- `Pile` : structure de donée représentant une pile d'objets quelconques (`Object`).

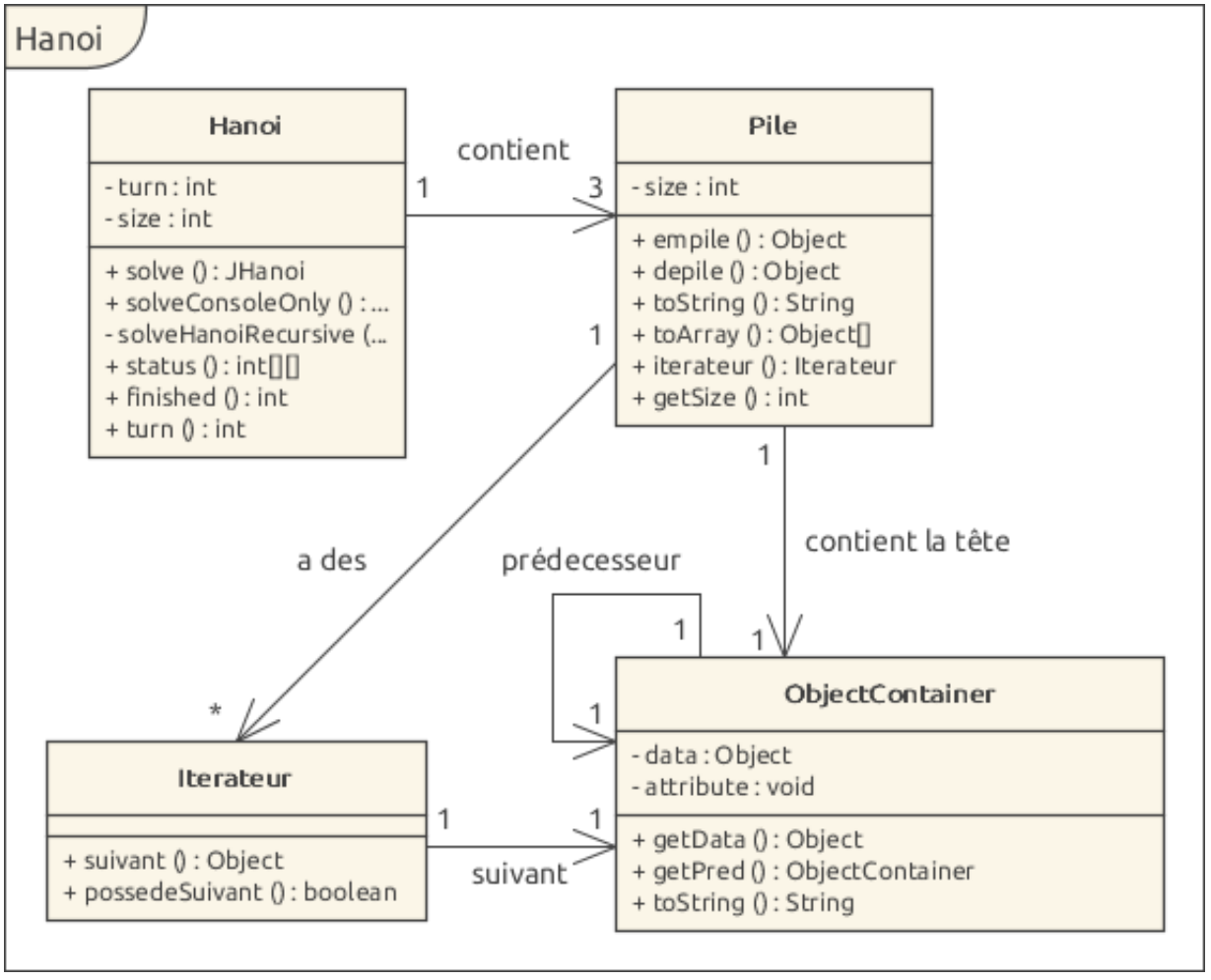
- `Iterateur` : un itérateur permettant de traverser une pile.
- `ObjectContainer` : un conteneur (*wrapper*) englobant un objet quelconque (`Object`) ainsi que le (`ObjectContainer`) précédant dans la pile-mère.

Programmes et Tests

Des classes dédiées aux tests ont été créées. Il s'agit des classes `TestHanoi` (permettant de résoudre le problème des tours de Hanoï en console), `TestHanoiGUI` (permettant de résoude le problème des tours de Hanoï avec l'interface graphique), et `TestPile` (qui contient des tests unitaires vérifiant le bon fonctionnement de la pile).
Ces trois classes contiennent une méthode `main` afin de pouvoir être lancée une à une, selon les besoins.

Schéma des classes

Voici le schéma UML des classes.



Note: un objet `JHanoi` est instancié au début du programme, en mode graphique. Cet objet va à son tour créer un

objet `Hanoi` .

Lancement du programme

Le programme `TestHanoi` peut être lancé avec un entier en paramètre optionel. Il permet de spécifier le nombre de disque de la tour (4 par défault).

Pour aller plus loin

Selon la légende, des moines déplacent sans arrêt de lourds disques de métal d'une aiguille à l'autre.

Question

En supposant des moines surentraînés capables de déplacer un disque à la seconde, combien de temps reste-t-il avant que l'univers disparaisse (celui-ci a actuellement 13,7 milliards d'années)?

Réponse

Le nombre de coups à jouer pour déplacer n disques est $2^n - 1$. Ceci peut être facilement montré par une preuve par induction et vérifié à la main pour des petits nombres. Intuitivement, il faut déplacer $n-1$ disques du départ à l'intermédiaire, le dernier disque du départ à l'arrivée, puis $n-1$ disques de l'intermédiaire à l'arrivée.

Donc itérativement:

- $K(0) = 0$
- $K(n) = 2\,K(n-1) + 1$

La formule générale est donc: $\boldsymbol{K(n) = 2^n - 1}$

Pour 64 disques à raison d'un mouvement par seconde, il faut donc $2^{64} - 1$ secondes, soit $1.8446744e+19$ secondes.

Dans un milliard d'années, il y a $10^{9'365'24'60'60} = 3.1536e+16$ secondes.

Il faudra donc *$(2^{64}$ - 1) / 3.1536e+16 = 584.94 milliards d'années* aux moins pour déplacer tous les disques.

En soustrayant les 13.7 milliards d'années s'étant déjà écoulés depuis le Big Bang, il reste donc a priori ***571.24 milliards d'années*** à l'univers, selon cette tradition.

Conclusion

Nous pouvons constaté que la résolution de ce problème est de plus en plus longue. En effet, sa complexité algorithmique optimale est en $O(2^n)$, ce qui est relativement lourd par rapport aux algorithme que nous avons l'habitude d'utiliser en informatique.

Annexes

Le code imprimé est annexé.

Sortie des programmes en console

TestPile

```
TEST: taille pile vide:0
TEST : vide au demarrage
TEST: empile/depile.
Empile:hello(1)
Empile:world(2)
Empile:my name is Bond (3)
Empile:James Bond (4)
TEST: taille pile plein:4
Depile:James Bond (4)
Depile:my name is Bond (3)
Depile:world(2)
Depile:hello(1)
TEST: Depile: previsible : Pile vide!
Empile:hello(1)
Empile:world(2)
```

```
Empile:my name is Bond (3)
Empile:James Bond (4)
TEST: ToString():[ <James Bond (4)> <my name is Bond (3)> <world(2)> <hello(1)> ]
TEST: foreach on ToArray()
James Bond (4)
my name is Bond (3)
world(2)
hello(1)
TEST Arrays.toString() on toArray()
[James Bond (4), my name is Bond (3), world(2), hello(1)]
TEST: Iterateur
PossedeSuivant()?:true
Suivant:James Bond (4)
PossedeSuivant()?:true
Suivant:my name is Bond (3)
PossedeSuivant()?:true
Suivant:world(2)
PossedeSuivant()?:true
Suivant:hello(1)
PossedeSuivant()?:false
```

TestHanoi

```
--- turn 0 ---
One:   [ <1> <2> <3> <4> ]
Two:   [ ]
Three: [ ]
--- turn 1 ---
One:   [ <2> <3> <4> ]
Two:   [ <1> ]
Three: [ ]
--- turn 2 ---
One:   [ <3> <4> ]
Two:   [ <1> ]
```

```
Three: [ <2> ]
--- turn 3 ---
One:   [ <3> <4> ]
Two:   [ ]
Three: [ <1> <2> ]
--- turn 4 ---
One:   [ <4> ]
Two:   [ <3> ]
Three: [ <1> <2> ]
--- turn 5 ---
One:   [ <1> <4> ]
Two:   [ <3> ]
Three: [ <2> ]
--- turn 6 ---
One:   [ <1> <4> ]
Two:   [ <2> <3> ]
Three: [ ]
--- turn 7 ---
One:   [ <4> ]
Two:   [ <1> <2> <3> ]
Three: [ ]
--- turn 8 ---
One:   [ ]
Two:   [ <1> <2> <3> ]
Three: [ <4> ]
--- turn 9 ---
One:   [ ]
Two:   [ <2> <3> ]
Three: [ <1> <4> ]
--- turn 10 ---
One:   [ <2> ]
Two:   [ <3> ]
Three: [ <1> <4> ]
--- turn 11 ---
One:   [ <1> <2> ]
```

```
Two:   [ <3> ]
Three: [ <4> ]
--- turn 12 ---
One:   [ <1> <2> ]
Two:   [ ]
Three: [ <3> <4> ]
--- turn 13 ---
One:   [ <2> ]
Two:   [ <1> ]
Three: [ <3> <4> ]
--- turn 14 ---
One:   [ ]
Two:   [ <1> ]
Three: [ <2> <3> <4> ]
--- turn 15 ---
One:   [ ]
Two:   [ ]
Three: [ <1> <2> <3> <4> ]
```