Notes de Cours INF 2010

Olivier Sirois

2018-01-01

Contents

0.1 Cours 1 - Intro

Important = courriel à tarkoo@gmail.com avec sujet

$$INF2010$$

avec la formation d'équipe. dans le courriel on met les noms et les matricules.

Les labos commencent la semaine du 15 Janvier. Les remises sont sur Moodle comme dab. 6 travaux pratique qui compte pour 5 % chacun. l'intra aura lieu le 19 février.

Les livres sont le Weiss et le Cormen (CLRS)

Le cours est en **Java**, on doit l'apprendre soit-même.

Chapter 1

1 - Intro + Complexités algorithmiques

Structure de données Une structure de donnée est une manière d'organisé les données pour qu'elles soient plus facile d'utilisation et en vue de les traiter de manière efficacement - Comme par exemples

- les traiter
- rechercher
- trier
- ajouts
- modifié
- accédés
- etc...

Définition officiel Structure lpogique servant à contenir des données dans le but de les traiter

Exemple : en ce qui concerne les nombres premier, il n'y a pas vraiment de manière de les calculer de manière directement, pour le faire, on doit énuméré tous les nombres et seulement conserver ceux qu'on peut vérifier comme étant premier en retirant les nombres ordinaires. Pour pouvoir vérifier si un nombre est premier, on fait une confirmation avec tous les éléments de notre liste de notre premier en les comparant avec la cible, si elle se trouve dans la liste elle est nécessairement un nombre premier.

lorsque n tend vers l'infini, nous avons environ 10% des nombres sont des nombres premiers.

- On peut définir une lecture séquentielle comme ayant une complexité de O(n). étant donné que nous avons vérifier un à un tous les nombres de la liste.

- On peut faire un arbre binaire pour représentés nos différents éléments de la liste. cela va nous permettre de plus facilement de voir ou est notre valeur. cela réduit à O(log(n)).
- On peut finalement faire un hash table, qui donne O(1) (1)

On représente normalement les algorithme en notation O. cette notation représente la tendance de l'algorithme lorsque le dataset tend vers $n = \infty$. C'est asymptotique.

Normalement, lorsqu'on imbrique des boucles for, c'est des n à la puissanc du nombre d'imbrications. Les récursivité et les boucles avec un nombre d'index qui se diminue amene des log(n). des fonction contentn aucune boucles donne des O(1)

1.1 Java

On choisit le Java car on ne veut généralement pas avoir à gérer la mémoire (sa peut être très inefficace en certains cas). Par contre, pour le prototypage d'algorithme, sa fait en sorte qu'on peut s'abstraire de certaines complexités. Par contre, l'utilisation de la JVM abstrait notre code de l'architecture de l'ordinateur étant donnée que la JVM est elle-même compilés pour chaque différentes architecture.

Chapter 2

Liste, Pile et File

Liste Tout simplement une collection d'objet arrangé de manière linéaire (1 $\rightarrow \infty$). On peut identifier chaque objet non seulement par sa valeur mais aussi par sa position dans la collection, autrement appelé indexe. On peut différencier deux type de liste en générale, soit:

- Liste par tableau: tout les objets sont placés concurentiellement en mémoire, l'indexe représente le décallage de mémoire entre la première position du tableau et celle que l'on désire accéder.
- Liste chainé : les objets ne sont pas placés concurentiellement, par contre, chaque objet de la liste contient un lien à l'objet suivant, qui sont chacun liés ensemble jusqu'au dernier élément de la liste, qui lien contient un lien à un NULL. La liste peut être soit chaîné dans une seule direction, mais elle peut aussi être chaîné dans les deux directions (doublement liés).

La liste nous sera utile pour implémenter des structure de données plus complexes (FIFO et LIFO), dictionnaire, etc. Le concept nous aidera à faire les premiers pas dans l'éthodes de la complexités algorithmique..

les actions possible sont normalement:

- add(): ajouter un élément
- remove(): enlever
- get(): obtiendre un élément
- set(): modifier un élément
- empty(): liste vide?
- size(): grosseur liste?

Normalement, lorsque notre liste n'est pas assez grosse pour contenire toutes les données (dans le cas d'un tableau), nous devons en recréer une autres avec une grosseur plus élévés. Normalement, on en fait une autre environ deux fois plus grosse pour ne pas à toujours avoir à allouer de la mémoire, seulement une fois de temps en temps.

Dans le cas de liste chaînée. Nous n'avons pas besoin vu que chaque élément est un élément en soi-même.

2.1 Classes et interfaces

On utilise l'interface **Collection** de Java pour pouvoir gérer nos liste. C'est très généralement et elle dérive **Iterable**.
INSERT SLIDE 9

Anytype est important, sa désigne n'importe quel type. C'est un template pour dire qu'on ne spécifie pas le type.

Il est possible d'effectuer une boucle for intelligente sur un objet de type Iterable. l'objet à sa propre interprétation de position, et avec les méthodes next() et hasnext() nous permet de passer à travers tous les objets de la boucle avec un while de manière très simple.

INSERT EXEMPLE WHILE ITERABLE slide 12

Note, un opération remove sur un itérateur de type while ne fonctionne pas bien. par que le remove change la liste et invalide l'itérateur en le faisant. Une modification peut être amené pour amener une solution.
Le static dans la slide 10 veut dire qu'on n'a pas besoin d'instancier l'objet pour utiliser. sa amène un découplage en la classe et sa méthode.

l'interface itérateur contient:

- bool hasNext();
- Anytype next();
- void remove();

itérable à une complexité de 0(n) sur array list et O(1) sur Linkedlist. C'est due au fait que sur le tableau il doit décaller toutes les éléments à l'insertion d'un élément tandis que sur le array list il doit seulement changer les liens. sa prend seulement quelques opérations.

2.1.1 interface List

l'interface **List** hérite de l'interface collection en plus d'hérité de itérable. sa se combine avec un interface ListIterator. La différence est que cette interface à le concept de position. Lorsqu'on fais un get, on doit utiliser une position pour pouvoir décrire l'objet que nous voulons avoir.

l'interface list à les complexités suivantes :

INSERT SLIDE 14

On utilise le linked liste pour nous sauver des multiples resize et ne pas avoir à mettre tous les objets de manière concurrentiel. On peut aussi avoir add/remove en O(1) si c'est au début ou à la fin, ce qui peut être intéressant dépendemment des applications.

Si le pointeur est toujours bien positionner dans la linked list, on peut réduire le add/remove en O(1). Comme par exemple si on fais des opérations sur tous les éléments de la liste.

ListIterator Un interface ayant un mélange de list et iterator.. sous-type de Iterator

INSERT SLIDE 18 - 19

On s'attend que les iterateurs fonctionnent de cette manière:

INSERT SLIDE 23

Pour les implémentations de liste chaîné et liste liées, svp regarder les figures du chap3. dans le Weiss.

2.1.2 Piles et Files

Les piles et files sont des structures supposément plus complexes que les listes. En fait, ce sont des liste, sauf que les insertions et les remove se font seulement dans une seule direction. dans le cas de la pile, on peut seulement ajouter et enlever au début de la liste. Tandis que la file on ajoute au début et enleve à la fin (ou vice-versa).

Applications:

- Stacks
  - Analyse d'expression (compilateur qui check si on met les fins de brackets)

  - évaluations post fixe (notations polish reverse)
  - modèles de computation (utiliser la stack pour caller des sous-routines en assembleur)

- Files
  - Buffers. (tampons)