

# IFT2015 :: hiver 2020

## Cours de structures de données, Université de Montréal

JANVIER 9, 2020 JANVIER 13, 2020 *by* CSUROS M

## Semaine 1: références et exercices

- [EXERCICES, RÉFÉRENCES](#)
- [LAISSER UN COMMENTAIRE](#)



## Références

### Type Abstrait de Données



- [Sedgewick & Wayne] §1.2 (<http://algs4.cs.princeton.edu/12oop/>). Il peut être utile de lire aussi §1.1 aussi qui donne une introduction accélérée au langage Java (dès 0!) et la convention de code dans le livre.



- [Tutoriel Java sur les collections](http://docs.oracle.com/javase/tutorial/collections/) (<http://docs.oracle.com/javase/tutorial/collections/>)






- [optionnel] techniques avancées: [Tutoriel Java sur les streams](https://docs.oracle.com/javase/tutorial/collections/streams/index.html) (<https://docs.oracle.com/javase/tutorial/collections/streams/index.html>).



- **Java** classes Java: Collection [[API doc](https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html) (<https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html>)], [AbstractCollection](http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/AbstractCollection.java) [[source](http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/AbstractCollection.java) (<http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/AbstractCollection.java>) OpenJDK 8], [ArrayList](http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/ArrayList.java) [[source](http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/ArrayList.java) (<http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/ArrayList.java>) OpenJDK 8]

### Appartenance-union



- o  [\(https://algs4.cs.princeton.edu/\)](https://algs4.cs.princeton.edu/) [Sedgewick & Wayne] §1.5 (<http://algs4.cs.princeton.edu/15uf/>) ou   [CLR] §21.1–21.3 (pour les curieux/ses, §21.4 contient la preuve crédit-débit du théorème sur union-par-rang avec compression de chemin)
- o Wikipédia: [relation d'équivalence](https://fr.wikipedia.org/wiki/Relation_d%27%C3%A9quivalence) ([https://fr.wikipedia.org/wiki/Relation\\_d%27%C3%A9quivalence](https://fr.wikipedia.org/wiki/Relation_d%27%C3%A9quivalence)), [logarithme itéré](https://fr.wikipedia.org/wiki/Logarithme_it%C3%A9r%C3%A9) ([https://fr.wikipedia.org/wiki/Logarithme\\_it%C3%A9r%C3%A9](https://fr.wikipedia.org/wiki/Logarithme_it%C3%A9r%C3%A9)), [fonction d'Ackermann](https://fr.wikipedia.org/wiki/Fonction_d%27Ackermann) ([https://fr.wikipedia.org/wiki/Fonction\\_d%27Ackermann](https://fr.wikipedia.org/wiki/Fonction_d%27Ackermann)), [hyperopérateurs](https://fr.wikipedia.org/wiki/Hyperop%C3%A9ration) (<https://fr.wikipedia.org/wiki/Hyperop%C3%A9ration>)
- o animations [Kevin Wayne]: [solution naïve \(QuickFind\)](https://www.cs.princeton.edu/courses/archive/fall12/cos226/demo/15DemoQuickFind.mov) (<https://www.cs.princeton.edu/courses/archive/fall12/cos226/demo/15DemoQuickFind.mov>), [solution efficace \(QuickUnion\)](https://www.cs.princeton.edu/courses/archive/fall12/cos226/demo/15DemoQuickUnion.mov) (<https://www.cs.princeton.edu/courses/archive/fall12/cos226/demo/15DemoQuickUnion.mov>)

## Exercices

# 1. Piles et queues

## 1.1 Pile impossible

Supposons qu'on exécute une séquence mixte d'opérations push et pop. Avec les push, on empile les éléments A, B, ..., J, dans cet ordre. À chaque pop on affiche la valeur retournée.

- o Quelles séquences sont possibles parmi les suivantes: EDCBAHGF IJ, EGIHFD C JAB, CFGHEI JDBA, DCBAEFGHJI?
- o Donner un algorithme pour décider si une séquence est possible.  
**Indice:** essayez de reconstruire les opérations dans le sens inverse (en «repilant» les éléments dans un passage de droit à gauche).
- o [difficile] Quel est le nombre exact de séquences possibles?

## 1.2 Demi-tour

On peut renverser l'ordre des éléments sur une file FIFO Q à l'aide d'une pile auxiliaire P:

```
Demi-tour(Q) // renverse la file Q
{
    P ← new Pile();      // initialiser une pile vide
    while (! Q.isEmpty())
        P.push(Q.dequeue());
    while (! P.isEmpty())
        Q.enqueue(P.pop());
}
```

Il est un peu moins évident comment renverser une file FIFO sans utiliser une structure de données auxiliaire.

- o Donner un tel algorithme: l'algorithme ne peut accéder à la file que par l'interface standard (opérations enqueue, dequeue et isEmpty). N'utiliser que des variables simples dans la solution (c'est à dire aucune structure auxiliaire dont la taille dépend de la taille de la file). Caractériser le temps de calcul en fonction du nombre des éléments

nombre des éléments.

**Indice:** cherchez une solution par récursion.

- Donner un algorithme pour renverser l'ordre des éléments dans une *pile*, sans aucune structure de données auxiliaire. N'utiliser que les fonctions de l'interface standard: `pop`, `push` et `isEmpty`.

## 1.3 Couper l'herbe sous le pied

Implanter une queue (avec opérations `enqueue(x)`, `dequeue()`) à l'aide d'une **pile**. (L'interface du TA pile supporte les opérations `push(x)`, `pop()` et `isEmpty()`, en un temps amorti borné par une constante.)

Analyser le temps de calcul et l'usage de mémoire des opérations en fonction de la taille de la file.

**Indice:** Il est clair qu'on peut implanter `enqueue` par `push`. Afin d'implanter `dequeue`, on a besoin d'accéder au fond de la pile. Développez donc un algorithme récursif pour supprimer un élément au fond de la pile.

## 2. Union-find

### MMORPG

Dans un certain jeu de rôle en ligne massivement multijoueur, on peut faire progresser son personnage dans des combats joueur-contre-joueur ou joueur-contre-monstre. Chaque joueur ou monstre  $x$  possède des **points de vie**  $v(x)$  et un **niveau** (ou classement)  $c(x)$ , tous les deux des entiers non-négatifs.

Un joueur commence avec  $v(x)=1$  et  $c(x)=0$ ; les monstres commencent avec un classement  $c(x)=c \geq 0$  arbitraire et une vitalité de  $v(x)=2^c$ . Les règles pour la mise à jour après combat sont comme suit: si  $x$  (joueur ou monstre) gagne contre  $y$  (joueur ou monstre), alors

- (a) le vainqueur reçoit les points de vie de son adversaire, et le perdant s'élimine du jeu (et donc ne peut plus jamais combattre):  

$$v(x) \leftarrow v(x) + v(y); v(y) \leftarrow 0$$
- (b) si  $x$  gagne contre un personnage de niveau supérieur, il s'élève au même niveau:  

$$c(x) \leftarrow c(y) \text{ si } c(x) < c(y)$$
- (c) si  $x$  gagne contre un adversaire de niveau égal, il avance un niveau:  

$$c(x) \leftarrow c(x) + 1 \text{ si } c(x) = c(y)$$
- (d) si  $x$  gagne contre un personnage de niveau inférieur, son classement ne change pas.

Démontrer formellement que  $c(x) \leq \lg v(x)$  à tout temps pour tout personnage et monstre vivant  $x$ . («Vivant»:  $v(x) > 0$ ;  $\lg$  dénote logarithme binaire.)

**Indice:** utiliser de l'induction sur le nombre de combats.

Publicités

