DM 3 - Algorithmes

Mise en place

Pour ce DM, commencez par forker le dépot git@gitlab.com:jrobert/dm_java_2.git , puis faites en un clone sur votre machine.

Remarque : si vous êtes à l'IUT, n'oubliez pas d'exporter les variables d'environnement pour le proxy au préalable :

```
export http_proxy="http://wwwcache.univ-orleans.fr:3128"
export https_proxy="http://wwwcache.univ-orleans.fr:3128"
```

Dans le dépots, vous trouverez l'architecture suivante

```
.
├── build
├── build.xml
├── src
│   └── BibDM.java
└── tests
    ├── ElementMinTest.java
    ├── IntersectionTest.java
    └── MinTest.java
```

Vous n'avez à intervenir que dans le dossier src.

Cette architecture permet de lancer des tests. Pour cela, mettez vous à la racine et tapez

```
ant test
```

Cela lance les tests et vous affiche ceux qui passent ou non. Pour avoir un rapport plus "joli", vous pouvez lancer

```
ant test-html
```

Cela crée un dossier reports avec des fichiers html de rapports sur les tests exécutés.

Exercices

Ce DM est composé de différents exercices, tous indépendants et dont la difficulté n'est pas croissante. Commencez par ce qui vous inspire le plus !

Chaque exercice correspond à une méthode à implémenter, mais vous pouvez / devez implémenter d'autres méthodes selon vos besoins.

Plus

Cet exercice vous permettra de voir si vous avez l'environnement permettant de travailler. Dans la classe BibDM, écrivez une méthode plus qui prend en entrée deux Integer et renvoie la somme des Integers.

Lancez les tests avant d'écrire le contenu de cette méthode et observez les rapports. Implémentez le code qui convient. Observez que les tests sont maintenant passés.

Autres Méthodes

Dans le fichier BibDM.java complétez les méthodes. Testez votre code avec `ant test-html`.

```java
import java.util.Collections;
import java.util.List;
import java.util.ArrayList;
public class BibDM{

  /**
   * Ajoute deux entiers
   * @param a le premier entier à ajouter
   * @param b le deuxieme entier à ajouter
   * @return la somme des deux entiers
   */
  public static int plus(int a, int b){
      return 0;
  }


  /**
   * Renvoie la valeur du plus petit élément d'une liste d'entiers
   * VOUS DEVEZ LA CODER SANS UTILISER COLLECTIONS.MIN (i.e. vous devez le faire avec un for)
   * @param liste
   * @return le plus petit élément de liste
   */
  public static Integer min(List<Integer> liste){
      return null;
  }


  /**
   * Teste si tous les élements d'une liste sont plus petits qu'une valeure donnée
   * @param valeur
   * @param liste
   * @return true si tous les elements de liste sont plus grands que valeur.
   */
  public static<T extends Comparable<? super T>> boolean plusPetitQueTous( T valeur , List<T> liste){
      return true;
  }



  /**
   * Intersection de deux listes données par ordre croissant.
   * @param liste1 une liste triée
   * @param liste2 une liste triée
   * @return une liste triée avec les éléments communs à liste1 et liste2
   */
  public static <T extends Comparable<? super T>> List<T> intersection(List<T> liste1, List<T> liste2){
      return null;
  }


  /**
   * Découpe un texte pour obtenir la liste des mots le composant. texte ne contient que des lettres de l'alphabet et des espaces.
   * @param texte une chaine de caractères
   * @return une liste de mots, correspondant aux mots de texte.
   */
  public static List<String> decoupe(String texte){
      return null;
  }


  /**
   * Renvoie le mot le plus présent dans un texte.
   * @param texte une chaine de caractères

   * @return le mot le plus présent dans le texte. En cas d'égalité, renvoyer le plus petit dans l'ordre alphabétique
   */

  public static String motMajoritaire(String texte){

      return null;
  }
```

```java
  /**
   * Permet de tester si une chaine est bien parenthesée
   * @param chaine une chaine de caractères composée de ( et de )
   * @return true si la chaine est bien parentjèsée et faux sinon. Par exemple ()) est mal parenthèsée et (())() est bien parenthèsée.
   */
  public static boolean bienParenthesee(String chaine){
      return true;
  }

  /**
   * Permet de tester si une chaine est bien parenthesée
   * @param chaine une chaine de caractères composée de (, de  ), de [ et de ]
   * @return true si la chaine est bien parentjèsée et faux sinon. Par exemple ([)] est mal parenthèsée alors ue ([]) est bien parenthèsée.
   */
  public static boolean bienParentheseeCrochets(String chaine){
      return true;

  }


  /**
   * Recherche par dichtomie d'un élément dans une liste triée
   * @param liste, une liste triée d'entiers
   * @param valeur un entier
   * @return true si l'entier appartient à la liste.
   */
  public static boolean rechercheDichotomique(List<Integer> liste, Integer valeur){
      return true;
  }



}
```