20/05/2021 - 8h30-10h30

Initiation à la Programmation (IP2)

Aucun document autorisé. Dans vos réponses n'utilisez **aucune des librairies de java** qui sont en rapport avec les listes, vous devez écrire tout ce qui vous est utile.

Vu l'état de la présentation de certaines copies du partiel, il faut rappeler ici que vous disposez de brouillon pour réfléchir, et que les réponses tiennent en quelques lignes seulement. Cela ne vous prendra pas beaucoup plus de temps de n'écrire au propre que lorsque vous avez les idées claires : si vous rendez quelque chose de confus, de peu lisible, le correcteur ne peut pas vous donner le bénéfice du doute.

Ce sujet est composé de 4 exercices. Ils sont indépendants, et apparaissent dans l'ordre que nous avons suivi pour la progression du cours. Le barème sur 21 est indicatif, il vous donne une idée du temps à consacrer aux exercices. S'il évolue légèrement ce sera pour donner de l'importance aux exercices traités complètement et correctement. Inutile donc de vous précipiter : si vous faites bien 3 des 4 exercices vous aurez certainement une meilleur note que si vous accumulez des erreurs.

Exercice 1 (5 points)

1. Ecrivez des méthodes qui permettent de résoudre les problèmes suivant :
   (a) (1.5 points) On cherche dans un tableau d'entiers t à afficher la plus grande valeur impaire qu'il contient. (Envisagez les cas limites)
   (b) (1.5 points) On a deux tableaux d'entiers $t1$ et $t2$ et on veut savoir combien d'éléments de $t1$ se trouvent également dans $t2$.
   (c) (1.5 points) On a deux tableaux de caractères t_a et t_b et on veut un nouveau tableau t_c obtenu en prenant alternativement le premier élément de t_a puis celui de t_b et ainsi de suite (précisez ce que vous décidez d'obtenir si t_a et t_b n'ont pas la même taille)
2. (0.5 point) Quelle est la valeur affichée par la séquence suivante

```
int rep=0;
for (int i=0;i<4;i++)
 for (int j=i; j<2*i; j++) rep++;
System.out.println(rep);
```

Exercice 2 (7.5 points) Après avoir autant expérimenté les cours enregistrés, vous conviendrez certainement que ce serait une bonne chose si on disposait d'un outil qui permettrait de débarrasser la bande sonore de tous les tics de langage qui s'y trouvent, c'est à dire d'effacer les occurrences des mots trop fréquents qui n'apportent rien au sens d'un texte. Typiquement ce sont les "heuuu", "bon", "alors", "ok", "hein" etc ... [1]

Nous allons considérer ici qu'un message se retrouve capté en mémoire sous la forme d'une liste simplement chaînée de caractères (chaque cellule ne contient qu'un seul caractère), et nous abordons par étapes le développement de cette fonctionnalité.

[1] d'ailleurs merci pour votre indulgence cette année

1. (2 points) Ecrivez une méthode build de la classe Message qui prend en argument une String et retourne le message correspondant en le stockant sous la forme d'une liste chaînée de caractères. Vous utiliserez ces deux classes :

```
public class Message {
 private Cellule tete;
}
public class Cellule {
 private char x;
 private Cellule next;
}
```

   Donnez à cette méthode la signature la plus appropriée, en la justifiant, et écrivez bien en détail tout ce dont vous avez besoin (constructeurs, méthodes auxiliaires, etc ...).

   On vous rappelle deux méthodes usuelles des String :

   - *char charAt(int index) returns the char value at the specified index*
   - *String substring(int beginIndex) returns a new string that is a substring of this string.*

2. On s'intéresse maintenant à la recherche d'un message dans un autre message, notre objectif étant de repérer un mot parasite dans un enregistrement. Nous procédons par étapes :

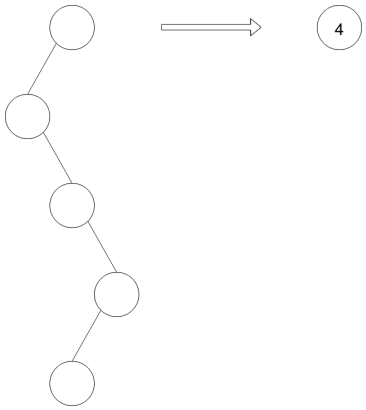
   (a) (1.5 points) Ecrivez une méthode boolean a_pour_prefixe(Cellule m), dans la classe Cellule qui répond true ssi le message commençant sur la cellule courante a pour préfixe le mot correspondant à m.

   (b) (1 point) Comme on souhaite ensuite l'effacer, nous allons maintenant récupérer la cellule qui précède une occurrence de ce mot. On se place dans le cadre où la cellule courante n'a pas pour préfixe m, et sous cette hypothèse qu'il n'est pas nécessaire de vérifier, on veut via une méthode Cellule get_begin_cut(Cellule m) chercher et retourner une cellule qui précède exactement une occurrence de m dans le message porté par la cellule courante. Ecrivez là.

3. (2 points) Ecrivez la méthode boolean delete_One(Message m) de la classe Message qui retire du message courant, si c'est possible, une seule occurrence de m. La méthode retourne si oui ou non le découpage a eu lieu. Un certain travail est à faire dans cette question, vous préciserez bien comment vous vous y prenez.

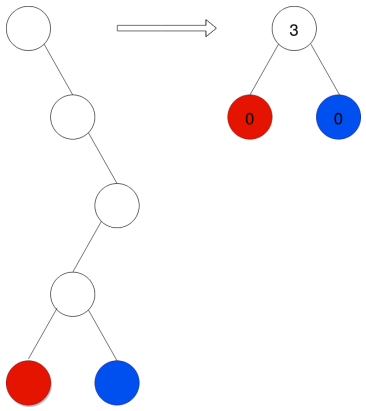
4. (1 point + éventuellement 0.5 bonus) Ecrivez une méthode clean de la classe Message qui retire d'un message courant toutes les occurrences de plusieurs mots donnés sous la forme de String.

Exercice 3 (5 points) On s'intéresse à un espace qui est un peu perdu lorsqu'on regarde de loin la structure des arbres. En effet, lorsque l'on a quelque part des branches qui consistent en un noeud suivit d'une suite de noeud n'ayant qu'un fils, on perd un peu de l'intérêt d'avoir un arbre binaire. C'est pourquoi on va vouloir réduire ces motifs à un seul noeud, celui qui sert de départ à ce type de branche, et on va y indiquer la longueur de la suite de noeuds non binaires qu'elle possédait.



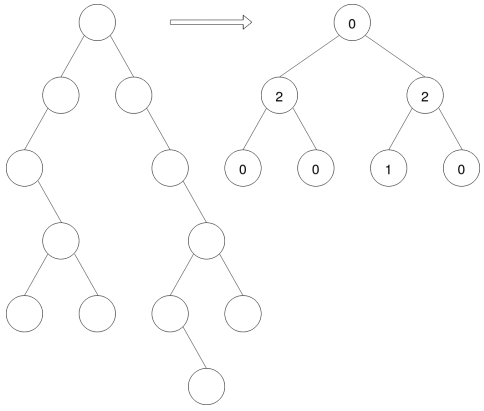
Dans cet exemple, à gauche vous avez la structure initiale, et à droite la forme "condensée" où le noeud restant "compte" pour 4 noeuds de plus.

Remarquez qu'on peut rencontrer ces motifs ailleurs que sur les extrémité de l'arbres. Dans ce cas on va remonter "l'extrémité utile", et à nouveau faire porter au noeud la taille de la branche supprimée. Voici un exemple :



Vous y voyez que les 2 noeuds les plus en bas sont remontés, et que le premier noeud comptabilise ceux de la branche supprimée (3).

Voici encore une dernière illustration, un peu plus générale qui devrait vous permettre de bien comprendre. C'est cette transformation qui est l'objet de cet exercice.



Nous travaillerons sur ces arbres :

```
public class Arbre {
  private Noeud racine;
}
public class Noeud {
  int val;
  private Noeud fils_g;
  private Noeud fils_d;
}
```

1. (1 point) Ecrivez une classe `Data` qui nous permettra de manipuler un objet dont les propriétés/attributs sont composés d'un entier et d'un noeud.

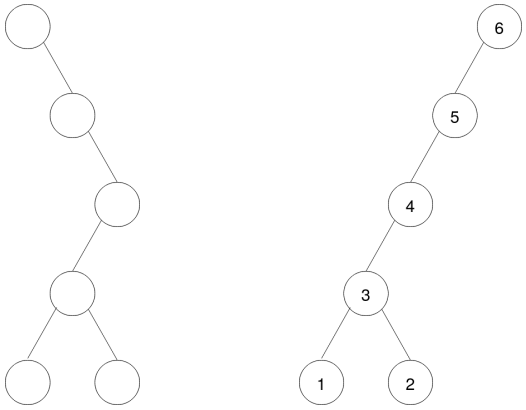
2. (2 points) Ecrivez une méthode `Data computeData()` de la classe `Noeud` qui retourne la taille de la branche issue de ce noeud, ainsi que sa terminaison utile, comme elles ont été présentées sur les exemples.

3. (2 points) Ecrivez les méthodes `void transforme()` dans la classe `Arbre` ainsi que sa jumelle dans la classe `Noeud` qui feront le travail suivant : chaque noeud va calculer la data qui est associée à la branche dont il est source, puis va localement "couper cette branche", c'est à dire intégrer les changement décrit par notre transformation, et enfin on poursuivra la transformation dans le même ordre qu'un parcours préfixe.

Exercice 4 (3.5 points) Dans l'exercice précédent on a choisi de "compresser" les branches non binaires. Dans celui là nous allons les "redresser" en privilégiant la direction "gauche". On souhaite également attribuer aux noeuds un nouveau numéro : celui qui correspond à l'ordre dans lequel on les rencontre dans un parcours suffixe.

Vous avez une explication graphique de ce que l'on souhaite obtenir sur les 2 exemples ci-dessous. Expliquez et écrivez votre solution pour ce problème.

transformation 1 :



transformation 2 :