TD Algorithmique - Feuille 3

Récursivité

1. **Inverse d'une chaîne**

   - Ecrire la méthode récursive "public String reverse(String s)" qui renvoie la chaine s écrite en sens inverse. Par exemple, si s = "bonjour", "reverse(s)" renvoie "ruojnob".
     Vous pourrez utiliser les méthodes "substring", "charAt" et le "+" de la classe "String".
     Exemple : String s1 = new String("je m'amuse"); String s2 = new String("beaucoup");
     s1.substring(3) renvoie "m'amuse"
     s1.charAt(3) renvoie "m"
     s1 + s2 renvoie "je m'amusebeaucoup"
   - Examiner la pile de récursion de l'exécution de la méthode "reverse(s)" avec s = "beau" (voir l'exemple de factorielle du cours). En déduire une version itérative de la méthode "reverse".

2. **Puissance**

   On veut calculer la puissance d'un nombre en utilisant seulement les opérations $+ - * \text{ et } /$.

   - En utilisant le schéma inductif usuel sur les entiers (0 est un entier, si n-1 est un entier, n est un entier), écrire la méthode récursive "public int puissanceRecursif(int x,int p)" qui renvoie x^p.
   - Pour mimiser la complexité du calcul, on propose d'utiliser la propriété suivante :
     si p est pair : $x^p = (x^2)^{p/2}$
     si p est impair : $x^p = x * (x^2)^{\frac{p-1}{2}}$
     Ecrire la méthode récursive "public int puissanceDichoRecursif(int x, int p)" qui renvoie x^p en utilisant la propriété précédente.

3. **Parenthèses**

   On considère les délimiteurs {, (, [, },),]. On veut vérifier qu'un texte est bien parenthésé. Par exemple, "oh! (dit-elle) (quelle [[belle]] journée)" est bien parenthésé. [Dommage (de travailler] !) est mal parenthésé.

   Compléter la méthode "private boolean correct(int i, String f)" de la classe "Parenthese" en annexe 1. Le principe est de passer dans f les délimiteurs ouvrants qui ont été rencontrés dans le texte jusqu'à l'indice $i-1$ (pour simplifier, on les écrits dans l'ordre inverse où on les a rencontrés). Par exemple, pour le texte "([bon{jour} à vous !))", quand on en est à l'indice $i = 6$ c'est à dire sur le caractère j, le paramètre f contient {[(.

4. **Relations de récurrence**

   Ecrire les relations de récurrence des méthodes des exercices 1, 2 et 3 et en déduire la complexité de ces méthodes.

5. **Au voleur !**

   Un voleur cherche à dévaliser un magasin. Comme il a mal au dos, il ne peut pas porter plus qu'un poids $Pmax$. Il cherche à emporter le butin de plus grande valeur.

   On considère la classe "Magasin" de l'annexe 2.

   - Quelle est la valeur du butin qu'il emportera pour un poids maximum de 30, 311 et 1000 ?
   - Compléter la méthode "private int volerAux(int poidsMax, int j)" de la classe $Magasin$.

6. **Le Sudoku**

```
|-------------------------|
||  .  .  .||  .  .  .||  .  .  .|
||  .  .  8||  6  .  3||  .  .  .|
||  .  .  1||  .  8  5||  .  .  .|
|-------------------------|
||  .  .  .||  .  .  .||  5  .  7|
||  .  .  .||  4  6  .||  .  1  .|
||  3  .  .||  5  .  7||  2  4  .|
|-------------------------|
||  5  .  .||  .  2  .||  .  .  1|
||  .  6  .||  .  .  .||  3  9  5|
||  .  9  .||  .  .  1||  6  8  .|
|-------------------------|
```

Remplissez la grille de sorte que chaque colonne, chaque ligne, chaque carré 3x3 contiennent une fois et une seule fois les chiffres de 1 à 9 (extrait de Nice Matin du 12/10/2006).

Compléter la méthode "placer" de la classe "Sudoku" en annexe 3 de façon à remplir la grille et renvoyer vrai s'il existe une solution, et renvoyer faux sinon.

Annexe 1 : les parenthèses

```
/** une classe pour vérifier si un texte est bien parenthésé */
public class Parenthese {
    private String texte; // le texte
    private int size; // la longueur du texte
    public Parenthese(String t) {
        texte = t;
        size = t.length();
    }
    /** renvoie vrai si le texte est bien parenthésé */
    public boolean correct() {
        return correct(0,"");
    }
    /** ANTECEDENT : f contient les délimiteurs ouvrants non refermés de texte[0..i-1]
     *  renvoie vrai si le texte de i à size :
     *  - contient les délimiteurs fermants correspondants à  f
     *  - est bien parenthésé  */
    private boolean correct(int i, String f) {
    // A COMPLETER
    }
    /** renvoie vrai si c est un délimiteur ouvrant */
    private boolean estOuvrant(char c) {
        return c=='(' || c=='[' || c=='{';
    }
    /** renvoie vrai si c est un délimiteur fermant */
    private boolean estFermant(char c) {
        return c==')' || c==']' || c=='}';
    }
    /** renvoie le délimiteur fermant qui correspond à c
     *  renvoie ' ' si c n'est pas un délimiteur ouvrant */
    private char ferme(char c) {
        char f=' ';
        if (estOuvrant(c)) {
            switch (c) {
            case '(' : f = ')'; break;
            case '[' : f = ']'; break;
            case '{' : f = '}'; break;
            }
        }
        return f;
    }
}
```

Annexe 2 : les voleurs

```
/** classe pour les articles d'un magasin */
public class Article {
  private int poids;
  private int valeur;
  private String nom;

  public Article(int p, int v, String n) {
      poids = p;
      valeur = v;
      nom = n;
 }
```

```
    public int poids() {
        return poids;
    }
    public int valeur() {
        return valeur;
    }
    public String toString() {
        return " nom : " + nom + "poids : " + poids + " valeur :" + valeur ;
    }
}

/** un magasin contenant des articles et  un voleur */
public class Magasin {
   Article[] magasin;   // les articles du magasin

    // un constructeur par défaut pour créé un magasin
   public Magasin() {
       magasin = new Article[8];
       magasin[0] = new Article(100,5000, "collier en or");
       magasin[1] = new Article(1000,1, "spaghetti");
       magasin[2] = new Article(100,5, "amandes décortiquées biologiques");
       magasin[3] = new Article(1,50, "saffran");
       magasin[4] = new Article(1000,1, "flageolets en conserve");
       magasin[5] = new Article(100,10, "faux filet");
       magasin[6] = new Article(10,2, "purée biologique");
       magasin[7] = new Article(300,10, "merguez");
   }
    /* renvoie le butin de plus grande valeur de poids maximum poidsMax */
    public int voler(int poidsMax) {
        return volerAux(poidsMax,magasin.length-1);
    }
    /* renvoie le butin de plus grande valeur de poids maximum poidsMax,
       pris dans les objets d'indice <= j */
    private int volerAux(int poidsMax, int j) {
       // A COMPLETER
    }
}
```

Annexe 3 : le Sudoku

```
/* une classe pour résoudre les sudoku de façon naive */
public class Sudoku{
    private int taille; // la taille de la grille
    private int tailleCarre; // la taille d'un carre dans la grille
    private int[][] grille; // la grille de sudoku

    public Sudoku() {
        this(9,3);
    }
    public Sudoku(int t, int tCarre) {
        taille = t;
        tailleCarre = tCarre;
        grille = new int[taille][taille];
    }
    /* place la valeur initiale v en (i,j) */
    public void init(int i, int j, int v) {
        grille[i][j]=v;
    }
```

```
/* CONSEQUENT : renvoie vrai si la  valeur v est possible pour la ligne l*/
private boolean lignePossible(int l,int v) {
    for (int j=0;j<taille;j++)
        if (grille[l][j]==v)return false;
    return true;
}
/* CONSEQUENT : renvoie vrai si la  valeur v est possible pour la colonne c */
private boolean colonnePossible(int c,int v) {
    for (int j=0;j<taille;j++)
        if (grille[j][c]==v)return false;
    return true;
}
/* CONSEQUENT : renvoie vrai si la valeur v est possible pour le carre
 * dont le coin (haut,gauche) est (i,j)*/
private boolean carrePossible(int i,int j,int v) {
    int coinI = (i/tailleCarre) * tailleCarre;
    int coinJ = (j/tailleCarre) * tailleCarre;
    for (int i1=0;i1<tailleCarre;i1++)
        for (int j1=0;j1<tailleCarre;j1++)
            if (grille[coinI+i1][coinJ+j1]==v) return false;
    return true;
}
/* renvoie l'indice de la  ligne de la case à droite de  (i,j) */
private int ligneSuivante(int i,int j){
    if (j==taille-1) return i+1;
    else return i;
}
/* renvoie la  colonne de la case à droite de  (i,j) */
private int colonneSuivante(int i,int j){
    if (j==taille-1) return 0;
    else return j+1;
}
/* CONSEQUENT : remplit la grille, renvoie false si c'est impossible */
private boolean remplir() {
    return placer(0,0);
}
/* CONSEQUENT : place une valeur en (i,j), renvoie false si c'est impossible */
private boolean placer(int i, int j) {
   // A COMPLETER
}
/* CONSEQUENT : renvoie une chaine qui represente la grille */
public String toString() {
    String result = "";
    for (int i=0;i<taille;i++) {
        if (i%tailleCarre==0) result  +=  ligne()+ "|\n";
        for (int j=0;j<taille;j++) {
            if (j%tailleCarre==0) result += "|| ";
            result += " " + grille[i][j];
        }
        result += "|\n";
    }
    result += "|" + ligne() + "|\n";
    return result;
}
public String ligne() {
    String result = "";
    for (int i=0;i<taille;i++) result += "---";
    return result;
}
```

```
    public static void main(String[] s) {
        Sudoku su = new Sudoku();
        su.init(1,2,8);
        su.init(1,3,6);
        su.init(1,5,3);
        su.init(2,2,1);
        su.init(2,4,8);
        su.init(2,5,5);
        su.init(3,6,5);
        su.init(3,8,7);
        su.init(4,3,4);
        su.init(4,4,6);
        su.init(4,7,1);
        su.init(5,0,3);
        su.init(5,3,5);
        su.init(5,5,7);
        su.init(5,6,2);
        su.init(5,7,4);
        su.init(6,0,5);
        su.init(6,4,2);
        su.init(6,8,1);
        su.init(7,1,6);
        su.init(7,6,3);
        su.init(7,7,9);
        su.init(7,8,5);
        su.init(8,1,9);
        su.init(8,5,1);
        su.init(8,6,6);
        su.init(8,7,8);
        System.out.println("possible : " + su.remplir());
        System.out.println(su);

//      Solution :
// ---------------------------|
// ||  6 3 5||  7 1 9||  8 2 4|
// ||  2 7 8||  6 4 3||  1 5 9|
// ||  9 4 1||  2 8 5||  7 3 6|
// ---------------------------|
// ||  4 2 9||  1 3 8||  5 6 7|
// ||  8 5 7||  4 6 2||  9 1 3|
// ||  3 1 6||  5 9 7||  2 4 8|
// ---------------------------|
// ||  5 8 3||  9 2 6||  4 7 1|
// ||  1 6 2||  8 7 4||  3 9 5|
// ||  7 9 4||  3 5 1||  6 8 2|
// |---------------------------|

    }
```