Université de Nice-Sophia Antipolis Ecole Polytech' Nice Sophia

SI1 et MAM1
2005–2006

Examen Mathématiques Discrètes
du 23 Janvier 2006

Durée: 2 heures

1	
2	
3	
4	
5	
6	

Tout document autorisé.

Toutes vos réponses doivent être justifiées par une démonstration ou un contre-exemple. Si vous pensez que le texte d'une question est ambigu (voire erroné) faites une hypothèse raisonnable et écrivez la sur votre copie.

1 Recurrence [3 points]

Montrer par récurrence que pour tout entier $n \geq 1$, la somme des carrés des n premiers nombres impairs est égale à $\frac{n(2n-1)(2n+1)}{3}$

On demande de montrer que $\sum_{i=1}^{n}(2i-1)^2 = \frac{n(2n-1)(2n+1)}{3}$.

Montrons le résultat par récurrence sur n

- Pour $n = 1$ c'est vrai.
- Supposons l'hypothèse vraie au rang n et montrons qu'elle est vrai au rang $n+1$:
  $\sum_{i=1}^{n+1}(2i-1)^2 = \sum_{i=1}^{n}(2i-1)^2 + (2n+1)^2 = \frac{n(2n-1)(2n+1)}{3} + (2n+1)^2 = \frac{(2n+1)(n+1)(2n+3)}{3}$

.....

2 Equation de récurrence [3 points]

La banque LVFF[1] propose une épargne rémunérée de la manière suivante: Si un client laisse une somme s sur son compte épargne durant une année complète, alors à la fin de l'année, le compte épargne du client est crédité de 10% d'intêret, plus une somme fixe de 1000 euros. Soit u_0, la somme déposée par un client à la fin de l'année A, et soit u_n l'argent disponible sur le compte épargne à la fin de l'année A+n en supposant qu'il n'a jamais retiré d'argent de son compte épargne. Déterminer et résoudre l'équation de récurrence vérifiée par la suite u_n.

L'équation de récurrence est $u_n = 1{,}1 * u_{n-1} + 1000$. C'est une équation de récurrence linéaire non homogène d'ordre un.

L'équation homogène associée admet pour solutions les suites de la forme $u_n = \alpha(1{,}1)^n$. L'équation non homogène admet une solution particulière constante, cette constante c vérifie c= 1,1c+1000, donc c=-10 000

L'ensemble des solutions de l'équation initiale est donc formé des suites de la forme $u_n = \alpha(1{,}1)^n - 10000$. D'après la condition initiale, on a $\alpha = u_0 + 10000$, et donc finalement $u_n = (u_0 + 10000) * (1{,}1)^n - 10000$.

1. eLle Va Faire Faillitte

. . . . . .

3 Une autre [4 points]

Résoudre l'équation de récurrence suivante :

$$u_n = 3 * u_{n-1} - 4u_{n-3} + 3^n \text{ avec } u_0 = \frac{27}{4}, u_1 = \frac{81}{4}, u_2 = \frac{3}{4}.$$

L'équation linéaire homogène associée est $u_n = 3 * u_{n-1} - 4u_{n-3}$. Son polynôme caractéristique $x^3 - 3x^2 + 4$ admet -1 comme racine simple et 2 comme racine double. Les solutions de l'équation linéaire homogène associée sont donc de la forme $u_n = \alpha(-1)^n + 2^n(\beta + \gamma)$. 3 n'étant pas racine du polynome caractéristique, l'équation non homogène admet une solution de la forme $u_n = c3^n$. En reportant dans l'équation de récurrence, on obtient $c = \frac{27}{4}$.

L'ensemble des solutions de l'équation non homogène est donc de la forme

$$u_n = \alpha(-1)^n + 2^n(\beta + n\gamma) + \frac{27}{4}3^n.$$

D'après les valeurs initiales de la suite , on a

$\alpha + \beta = 0$

$-\alpha + 2\beta + 2\gamma = 0$

$\alpha + 4\beta + 8\gamma + 60 = 0$

D'où $\alpha = -\frac{20}{3}$; $\beta = \frac{20}{3}$ et $\gamma = -10$.

. . . . . .

4 Code récursif [5 points]

Soit A un alphabet fini, muni d'un ordre total $\leq$. On considère A^* l'ensemble des mots finis sur A

Soit m un mot de longeur l. Pour tout i, $1 \leq i \leq l$, on note m_i la ième lettre de m.

On dit qu'une mot m de longueur l est décroissant, si il est vide ou si pour tout i ,$1 \leq i \leq l-1$ $m_{i+1} \leq m_i$

Si de plus il n'y a pas deux lettres égales dans le mot m, m est dit strictement décroissant.

Soit A^*_D le sous ensemble de A^* constitué des mots décroissants, et A^*_{SD} le sous-ensemble de A^*_D constitué des mots strictement décroissants.

Donner une définition inductive et un code récursif pour les fonctions suivantes et déterminer la complexité de chacune de ces fonctions [2].

1. Decroissant : $A^* \rightarrow \{vrai, faux\}$
   Cette fonction retourne vrai si et seulement si le mot considéré est décroissant
2. Simplifie : $A^*_D \rightarrow A^*_{SD}$
   Cette fonction retourne le mot de A^*_{SD} ayant le même ensemble de lettres que le mot initial, mais où les répétitions de lettres ont été éliminées .
3. Intersection : $A^*_{SD} \times A^*_{SD} \rightarrow A^*_{SD}$
   Cette fonction retourne le mot de A^*_{SD} contenant les lettres qui sont dans les deux mots.

---

1. Decroissant
   - Decroissant(ϵ)= vrai
   - pour toute lettre a, Decroissant (a) = vrai
   - pour toute lettre a et tout mot m, Decroissant (a.m)= Decroissant (m) et a $\geq$ m.premiereLettre()

2. on pourra utiliser les méthodes premiereLettre, finMot, estVide, etc. vues en TD et les supposer en temps constant

- On considère les méthodes utiliées et celles qui sont demandées comme des méthodes d'instance appartenant à une classe `Mot`

```
public boolean decroissant(){
  if ( this.estVide() ){  return true;}
  Mot finMot = this.finMot();
  if ( finMot.estVide() ){  return true;}
  return ( this.premiereLettre() >= finMot.premiereLettre()) && finMot.decroissant();
}
```

- Si n est la longueur du mot , la complexité vérifie $c_n = c_{n-1} + 1$ et est linéraire en la longueur du mot traité (c'est-à-dire en $\Theta(m.length())$).

2. Simplifie
   - Simplifie(ϵ) = ϵ
   - pour toute lettre a, Simplifie (a)=a
   - Si m.premierelettre()=a, Simplifie (am) = Simplifie (m)
   - Si m.premierelettre()<>a, Simplifie (am) = a.Simplifie (m)

```
public Mot simplifie(){
  if ( this.estVide() ){  return motVide() ;}
  Mot finMot = this.finMot();
  if ( finMot.estVide() ){  return new Mot(this) ;}
  if ( this.premiereLettre() != finMot.premiereLettre() )
   {return this.premiereLettre().toMot().concat(finMot.simplifie()) ; }
  return finMot.simplifie() ;
}
```

   - La complexité est linéraire en la longueur du mot traité.

   Intersection
   - Intersection (m, ϵ)=Intersection (ϵ, m)=ϵ
   - Intersection (am, am') a. Intersection(m,m')
   - Si a < a' Intersection(am,a'm')=Intersection(am,m')
   - Si a'<a Intersectin (am, a'm')=Intersection (m,a'm')
   -
```
public Mot intersection(Mot m){
  if ( this.estVide() ){  return new Mot(m) ;}
  if ( m.estVide() ( return new Mot(this) ;)
  if ( this.premiereLettre() = m.premiereLettre() ){
     return  this.premiereLettre().toMot().concat(this.finMot().intersection(m.finMot()) ;}
  if ( this.premiereLettre() < m.premiereLettre() ) {return this.intersection (m.finMot() }
  return this.finMot().intersection(m) ;
}
```

   - La complexité est proportionnelle à la somme des longueurs des mots.

.....

5 Les flocons de Koch [5 points]

Pour construire un flocon de Koch , on démarre avec un triangle équilatéral dont chacun des cotés représente un segment. On découpe chaque segment en trois segments égaux et on remplace celui du milieu par deux segments qui forment avec le segment remplacé un triangle équilatéral. Ce processus est répété récursivement.

On obtient des figures successives comme suit :

Voici une applet Java permettant de dessiner de tels flocons (dans la suite on s'intéressera à la méthode `drawcurve`) :

```
import java.applet.*;
import java.awt.*;

public class Koch extends Applet {
    int level = 1;

    public void init() {
        setBackground(new Color(240,248,255));
    }

    public boolean mouseDown(Event ev, int x, int y) {
        if (!ev.metaDown()) level += 1;
        else if (level>1) level -= 1;
        repaint();
        return true;
    }

    public void paint(Graphics g) {
        // trois courbes, en tournant chaque fois de 120 degrés
        double x1=10, y1=310, angle=-Math.PI/3;
        for (int i=0; i<3; i++) {
            drawCurve(g,x1,y1,angle,350,level);
            x1 += 350*Math.cos(angle);
            y1 += 350*Math.sin(angle);
            angle += 2*Math.PI/3;
        }
    }

    private void drawCurve(Graphics g, double x1, double y1,
        double angle1, double sideLength, int level) {
    // (x1,y1), (x2,y2), (x3,y3), (x4,y4) = Points de départ des segments
    // sideLength = longueur de chaque segment
       double x2, y2, angle2, x3, y3, angle3, x4, y4;
       if (level>1) {
         sideLength /= 3;
         level -= 1;
         drawCurve(g, x1,y1, angle1, sideLength, level);
         x2 = x1+sideLength*Math.cos(angle1);
         y2 = y1+sideLength*Math.sin(angle1);
         angle2 = angle1-Math.PI/3;
         drawCurve(g, x2,y2, angle2, sideLength, level);
         x3 = x2+sideLength*Math.cos(angle2);
         y3 = y2+sideLength*Math.sin(angle2);
         angle3 = angle1+Math.PI/3;
         drawCurve(g, x3,y3, angle3, sideLength, level);
         x4 = x3+sideLength*Math.cos(angle3);
         y4 = y3+sideLength*Math.sin(angle3);
         drawCurve(g, x4,y4, angle1, sideLength, level);
      }
   else {
```

```
      g.drawLine((int)x1,(int)y1,
          (int)(x1+sideLength*Math.cos(angle1)),(int)(y1+sideLength*Math.sin(angle1)));
      }
   }
}
```

1. Donnez le nombre d'appels de la méthode `g.drawLine` en fonction de la valeur `level`=n lors de l'appel de `drawCurve`.
2. Donnez la longueur du flocon obtenu, en fonction de la longueur L d'un coté du triangle initial et de n.
3. Donnez la surface du flocon obtenu, en fonction de la surface S du triangle initial et de n.

6 Si vous avez fini le reste... [4 points]

Montrer que le produit de n entiers consecutifs quelconque est divisible par n!

Une solution possible :
Posons $P(k,n) = \prod_{i=0}^{n-1}(k+i)$
On montre que $P(k+1,n) = P(k,n) + n * P(n-1,k+1)$.
On conclut par récurrence sur les couples d'entiers.
. . . . . .

Espace pour débordement