IUT Arles– Info 1ère année - Matière AP (Module Algorithmique)	TD 9 Algorithmique

Révisions

Exercice 1 : Jeu de pierre-papier-ciseaux

Ecrire un algorithme permettant de jouer au jeu pierre-papier-ciseaux contre l'ordinateur. L'utilisateur et l'ordinateur ont trois coups possibles : pierre, papier ou ciseaux. Les règles sont les suivantes :

- les deux joueurs jouent la même chose : pas de point
- Pierre contre Papier : Papier gagne (et marque 1 point)
- Pierre contre Ciseaux : Pierre gagne (et marque 1 point)
- Ciseaux contre Papier : Ciseaux gagne (et marque 1 point)

L'utilisateur entre 1, 2 ou 3 et l'ordinateur tire un nombre aléatoire compris entre 1 et 3. On affiche à chaque fois le coup correspondant au nombre choisi par chaque joueur (1=pierre, 2=papier, 3=ciseaux) ainsi que le gagnant.
La partie est remportée par le premier à atteindre 10 points.

On utilisera la fonction aléatoire pour obtenir une valeur aléatoire :
Fonction Aléatoire (entier nb) : entier, Renvoie une valeur tirée aléatoirement entre 0 et Nb.
Par exemple, si Nb ← 5, Aléatoire renvoie un résultat compris entre 0 et 5.
Exemple de programme

```
Variable Nb, Resultat : numérique
Nb ← 12
Résultat ← Aléatoire (Nb)
Ecrire Résultat
```

```
Variable choix_joueur, choix_ordinateur, score_joueur, score_ordinateur,
score_max : numériques

score_max ← 10
score_joueur ← 0
score_ordinateur ← 0;
repéter
      Ecrire « 1 : Pierre" »
      Ecrire « 2 : Papier »
      Ecrire « 3 : Ciseaux »
      Ecrire « Votre choix : »
      Lire choix_joueur

      choix_ordinateur = 1+ Aléatoire(2)
      Selon cas
        choix_joueur = 1 :
            // Pierre
            Selon cas
              choix_ordinateur = 1 :
                  // Pierre
                  Ecrire « (vous) pierre | pierre (ordinateur) »
                  Ecrire « Coup nul »
              choix_ordinateur = 2 :
                  // Papier
                  Ecrire «  (vous) pierre | papier (ordinateur) »
                  Ecrire « L'ordinateur gagne »
                  score_ordinateur = score_ordinateur + 1
```

```
            choix_ordinateur = 3 :
                // Ciseaux
                Ecrire « (vous) pierre | ciseaux (ordinateur) »
                Ecrire "Vous gagnez" »
                score_joueur = score_joueur + 1;
        Fin Selon
    choix_joueur = 2:
        // Papier
        Selon cas
          choix_ordinateur = 1 :
              // Pierre
              Ecrire « (vous) papier | pierre (ordinateur) »
              Ecrire « Vous gagnez »
              score_joueur = score_joueur + 1;

          choix_ordinateur = 2 :
              // Papier
              Ecrire « (vous) papier | papier (ordinateur)") »
              Ecrire « Coup nul »

          Choix_ordinateur = 3 :
              // Ciseaux
              Ecrire « (vous) papier | ciseaux (ordinateur) »
              Ecrire «  L'ordinateur gagne »
              score_ordinateur = score_ordinateur + 1;
        Fin selon
    choix_joueur = 3 :
        // Ciseaux
        Selon cas
          choix_ordinateur = 1 :
              // Pierre
              Ecrire « (vous) ciseaux | pierre (ordinateur) »
              Ecrire « L'ordinateur gagne »
              score_ordinateur = score_ordinateur + 1;

          choix_ordinateur = 2 :
              // Papier
              Ecrire « (vous) ciseaux | papier (ordinateur) »
              Ecrire « Vous gagnez »
              score_joueur = score_joueur + 1;

          choix_ordinateur = 3 :
              // Ciseaux
              Ecrire « (vous) ciseaux | ciseaux (ordinateur) »
              Ecrire « Coup nul »
        Fin selon
  Sinon
        Ecrire « Ce n'est pas un choix valable. »
  Fin Selon
  Ecrire « Joueur: ", score_joueur, " | Ordinateur : ",score_ordinateur
Tant que (score_joueur<score_max et score_ordinateur < score_max)
Si ( score_joueur = score_max ) alors
  Ecrire « Bravo, vous avez gagne la partie !!! »
Sinon
  Ecrire « Vous avez perdu la partie !!! »
```

Il est aussi possible de ne tester que l'égalité et les cas gagnants, le reste amenant à la perte de la manche

Exercice II : Ecrire une fonction dont le résultat, booléen, indique si un nombre entier est premier ou non. Pour cela, essayer de diviser le nombre par tous ceux qui lui sont inférieurs, puis seulement par ceux qui sont nécessaires.
Remarque : On doit créer une fonction Modulo. On peut pour cela utiliser la condition (n/d)*d=n pour tester si d divise n.

1- créer la fonction Modulo (→a : entier, → b entier) : entier qui renvoie le reste de la division de a par b. Vous pouvez utiliser l'opération division entière.
2- Créer les versions successives suivantes de la fonction Premier
   a. Version 1 : on essaie les divisions par 2, 3, …, n-1
   b. Version 2 : On s'arrête au premier diviseur
   c. Version 3 : On s'arrête au premier diviseur ou à √n
   d. Version 4 : On pense à ne pas essayer 4, 6, 8, …
   e. (en Option) : Version 5 : On pense à ne pas essayer les multiples de 2, ni de 3. On essaie donc seulement 2, 3, 5, 7, 11, 13, 17, 19, 23, 25, … Soit 2, 3 puis des nombres de la forme 6 x k – 1 et 6 x k + 1. On passe de 6 x k – 1 à 6 x k +1 en ajoutant 2, puis de 6 x k + 1 à 6 x (k+1) – 1 en ajoutant 4.
   f. Version 6 (en option) : Pour accélérer encore la recherche, on n'essaie que les diviseurs premiers, stockés à l'avance, dans un tableau TabPrem qui contient donc les NombreD nombres premiers inférieurs à la racine carrée de Maxint.

```
// Si l'on ne dispose pas de l'opérateur modulo, on commence par créer une
fonction Modulo.
Cette fonction renvoie le reste s'il existe. Elle renvoie -1 en cas
d'erreur.
fonction Modulo (→a : entier, → b entier) : entier
variable q : entier
Début
      Si b = 0 alors
            Résultat -1
      Sinon
            q ← a / b   // division entière
            // on est sur de ne jamais l'appeler avec b = 0,
            // sinon on renvoie une erreur
            Résultat a – b * q
      Fsi
Fin

a = b * q + r ; donc pour avoir le reste il suffit de faire a – b * q


Version 1 : on essaie les divisions par 2, 3, …, n-1
Fonction Premier (→n : entier) : booleen
Variables Prem : booleen
Variable d : entier
Début
      Prem ← Vrai
      Répéter pour d=2 à n-1
            Si Modulo(n, d) = 0 alors
                  Prem ← Faux
            Fsi
      FinPour
      Resultat Prem
Fin


// Version 2 : On s'arrête au premier diviseur
d ← 2
```

```
Prem ← Vrai
Tant que (Prem = Vrai et d < n ) faire
      Si Modulo (n,d)=0 alors
            Prem ← Faux
      Sinon
            d ← d +1
      Fsi
FinTantQue


// Version 3 : On s'arrête au premier diviseur ou à √n
dmax ← PartieEntière(RacineCarrée(n)) // pour ne pas recalculer √n
d ← 2
Prem ← Vrai
Tant que (Prem = Vrai et d < dmax) faire
      Si Modulo(n,d) = 0 alors
            Prem ← Faux
      Sinon
            d ← d + 1
      Fsi
FinTantQue


// Version 4 : On pense à ne pas essayer 4, 6, 8, …
dmax ← PartieEntiere(RacineCarrée(n)) // pour ne pas recalculer √n
Si Modulo (n,2) = 0 alors
      Prem ← Faux
Sinon
      Prem ← Vrai
Fsi
d ← 3
Tant Que Prem = Vrai et d <dmax faire
      Si Modulo(n,d)=0 alors
            Prem ← Faux
      Sinon
            d ← d + 2
      Fsi
Fin Tant Que


// Version 5 : On pense à ne pas essayer les multiples de 2, ni de 3
// On essaie donc seulement 2, 3, 5, 7, 11, 13, 17, 19, 23, 25, …
// Soit 2, 3 puis des nombres de la forme 6 x k - 1 et 6 x k + 1
// On passe de 6 x k - 1 à 6 x k +1 en ajoutant 2
// puis de 6 x k + 1 à 6 x (k+1) - 1 en ajoutant 4
dmax ← PartieEntiere(RacineCarrée(n))
d ← 5
Pas ← 2
Si (Modulo (n, 2) =0 ou Modulo(n,3) = 0) alors
      Prem ← Faux
Sinon
      Prem ← Vrai
Fsi
Tant Que (Prem = Vrai) et (d <dmax) faire
      Si Modulo(n,d)=0 alors
            Prem ← Faux
      Sinon
            d ← d + pas
            pas ← 6 - pas
      Fsi
Fin Tant Que
```

On n'élimine pas, parmi les diviseurs à essayer, les multiples de 7, de 11, …, de manière analogue, car le gain sur les calculs ne compense pas la complication apportée à l'algo.

Version 6 : Pour accélérer encore la recherche, on n'essaie que les diviseurs premiers, stockés à l'avance, dans un tableau TabPrem qui contient donc les NombreD nombres premiers inférieurs à la racine carrée de Maxint.

```
d ← 1
Prem ← Vrai
Tant Que (Prem= Vrai et d <= NombreD) faire
      Si Modulo (n, TabPrem[d])=0 alors
            Prem ← Faux
      Sinon
            d ← d+1
      Fsi
FinTantQue
```

Exercice III : Afficher la décomposition d'un nombre en produit facteurs premiers.
Méthode : Si le nombre n admet le facteur premier p, il s'écrit n = p x m, il suffit alors d'écrire p , puis de recommencer avec m.

```
Variables N, d : entier
Ecrire(« Nombre à décomposer ? »)
Lire (N)
d ← 2   // diviseur premier à essayer
Tant Que N > 1 faire
      Si Modulo (N, d) = 0 alors
            Ecrire (d)
            N ← N / d
      Sinon
            d ← d + 1
      Fsi
FinTantQue
```

Pour bien comprendre cet algorithme, il faut remarquer que lorsque d n'est pas un nombre premier, N n'est pas divisible par d car on a déjà divisé N par les facteurs premiers de d.
On peut éviter d'essayer tous les entiers à partir de 2, mais cela complique l'algorithme : on commencera par extraire tous les deux, puis, dans une seconde boucle, les autres facteurs premiers à, partir de 3, essayant successivement 3, 5, 7, etc, …
Exemple : a= 50 et b= 25
Donc d prend la valeur 25

Exercice IV : Ecrire la fonction pgcd, dont le résultat est le plus grand commun diviseur des deux nombres fournis en paramètres. Une première idée consiste à essayer, en descendant, tous les pgcd éventuels jusqu'à atteindre un diviseur commun aux deux nombres. Plus efficace est l'algorithme d'Euclide, basé sur la propriété pgcd(a,b) = pgcd (b, a modulo b), pour a et b > 0.

Dans la première méthode, le mieux est d'initialiser le diviseur commun éventuel au plus petit des deux nombres (supposés tous les deux > 0), car le PGCD lui est inférieur au égal. On n'a pas à s'inquiéter de l'arrêt de la boucle : au pire elle s'arrêtera pour d = 1 (a et b premiers entre eux).
Exemple pgcd(27,9) on trouve 9
Pgcd (25, 10) : On trouve 5

```
Fonction PGCD(→a : entier, → b : entier) : entier
Variable d : entier
Début
      Si (a < b) alors
            d ← a
      Sinon
            d ← b
      Fsi
      Tant que (non(modulo(a,d)=0) et modulo (b, d)=0) faire
            d ← d -1
      Fin Tant Que

      Résultat d
Fin

Deuxième méthode : Avec l'algorithme d'Euclide
Fonction PGCDEuclide(→a : entier, → b : entier) : entier
Variable r : entier  // reste de la division de a par b
Début
      Tant que (b>0) faire
            r ← modulo(a,b)
            a ← b
            b ← r
      Fin Tant Que

      Résultat a
Fin
Le résultat est le dernier reste non nul. Soit p le pgcd de a et b, il est
obtenu dans une exécution de la boucle qui calcule r ←p, a←b et b ←p. On
a alors encore b>0, donc une nouvelle exécution de la boucle est effectuée
r←0, a ←p, b←0 : la boucle s'arrête enfin, mais p est dans a.
Contrairement à une idée répandue, il n'est pas nécessaire, lorsque a < b,
de procéder à l'échange de leurs valeurs, avant la boucle. En effet, si a <
b , la boucle effectue les affectations suivantes : r ← a (car a < b, a
modulo b = a), puis a ← b, enfin b ← r, ce qui échange a et b.
```

Exercice V : Ecrire la fonction ppcm, en utilisant la fonction pgcd (rappel pgcd(a,b) * ppcm (a,b) = a x b).

Exercice VI : La conjecture de Goldbach veut que tout nombre pair strictement supérieur à 2 soit la somme de deux nombres premiers. Ecrire un programme qui permet de vérifier cette conjecture. Cet entier sera passé en paramètre à une fonction qui calculera et affichera les deux nombres premiers dont la somme produit cet entier (s'il y a plusieurs paires, les afficher).

```
Procédure Générer_Couple (→ m : numérique)
Début
Variable a, b : numériques
      Répéter Pour a = 1 à m/2
            b ← m – a
            Ecrire «(», a, « , », b, « ) »
            Si (Premier (a) et Premier(b)) alors
Ecrire « "Conjecture de Goldbach vérifiée car », m, « = », a, « + », b
            Fsi
      FinPour
Fin
```

```
Dans le programme principal, il faut verifier que le nombre inséré soit
pair (Nb modulo 2 = 0)
```

Exercice VII : Deux nombres entiers N et M sont dits amicaux si la somme des diviseurs de N (N non compris) vaut M et la somme des diviseurs de M (M non compris) vaut N.

- Ecrire une fonction retournant la somme des diviseurs d'un entier passé en paramètre :
- Ecrire une fonction recevant en paramètre deux entiers et retournant une valeur booléenne selon qu'ils sont amicaux ou non.

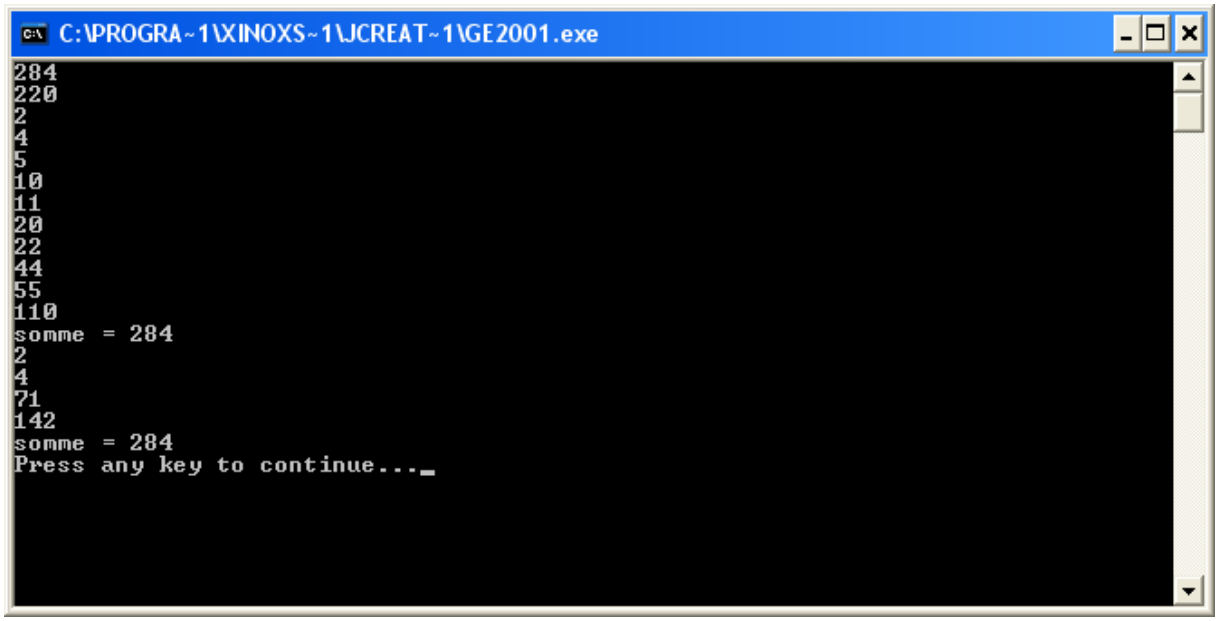
Ex : 220 et 284, 1.184 et 1.210, 17.296 et 18.416, 9.363.584 et 9.437.056.
Les nombres amicaux ont une histoire liée depuis longtemps à la magie et à l'astrologie. Par exemple, certains commentateurs juifs de la Genèse pensaient que Jacob avait donné 220 moutons (200 femelles et 20 males) à son frère quand il commença à craindre que son frère le tue (Genèsé 32:14). Le philosophe Iamblichus de Chalcis (ca. 250-330 A.C.) écrit que les pythagoriciens connaissent ces nombres qu'ils appellent amicaux et leur associent certaines qualités sociales (comme 220 et 284) et Pythagore aurait parlé d'un ami qui 'était un autre lui' comme le sont 220 et 284.
Il n'existe pas de formule ou méthode connues pour déterminer les nombres amicaux mais au fil des ans, certains types spéciaux ont été découverts. Thabit ibn Kurrah (ca.850 A.C.) nota que:
si n>1 et si $p = 3.2^{n-1}-1$, $q = 3.2^{n}-1$ et $r = 9.2^{2n-1}-1$ sont premiers,
alors $2^{n}pq$ et $2^{n}r$ sont des nombres amicaux.
Il fallut cependant des siècles pour que cette formule produise les $2^{ème}$ et $3^{ème}$ paires de nombres amicaux! Fermat annonça la paire 17.296 - 18.416 (n=4) dans une lettre à Mersenne en 1636. Descartes écrivit à Mersenne en 1638 pour lui signaler la paire 9.363.584 - 9.437.056 (n=7). Euler ajouta quant à lui une liste de 64 nouveaux nombres amicaux en faisant ainsi 2 erreurs qui furent découvertes en 1909 et 1914. En 1866 un jeune garçon de 16 ans, Nicolo Paganini, découvrit la paire 1.184 - 1.210 qui avaient été ignorée jusque là.

Des recherches par ordinateur ont permis de trouver toutes les paires de nombres amicaux de moins de 10 chiffres ainsi quelques autres encore plus grands pour en arriver à un total de 7.500 paires. On n'a pas pu montrer ni qu'il existe un nombre infini de paires ni qu'il existe une paire de nombres premiers entre eux. Si une telle paire existe, chacun des nombres doit comporter plus de 15 chiffres et leur produit doit doit être divisible par au moins 22 nombres premiers.

```
Variables M, N, SM, i : numériques
Ecrire "Introduisez les nombres à tester:"
Ecrire "N="
Lire N
Ecrire "M="
Lire M
SN ← 1
Répéter pour i=2 à N-1 faire
      si N mod i=0 alors
            SN ← SN+i
      fsi
fpour

SM ← 1
Répéter Pour i=2 à M-1 faire
      si M mod i=0 alors
            SM ← SM+i
      fsi
Fpour
```

```
si (SN=M) et (SM=N) alors
      Ecrire N, " et ", M," sont des nombres amicaux!"
sinon
      Ecrire N, " et ", M," ne sont pas des nombres amicaux!"
fsi
```

```
public class amicaux
{

        public static void main (String [] arg)
        {

int N, M, SM, SN, i;
N = Lire.i();
M = Lire.i();
        SM = 1;
        for (i=2; i<M;i++)
        {
                if ((M%i)==0)
                {
                                SM = SM +i;
                                System.out.println(i+" ");
                }

        }
        System.out.println("somme = "+SM);
        SN = 1;
        for (i=2; i<N;i++)
        {
                if ((N%i)==0)
                {
                                SN = SN +i;
                                System.out.println(i+" ");
                }

        }
        System.out.println("somme = "+SM);




        }       // main
        //--------------------------------------------------------------------------//
```



Autre solution :

```
public class amicaux
{

	public static void main (String [] arg)
	{

int N, M, SM, SN, i;
N = Lire.i();
M = Lire.i();
	SM = 1;
	i = 2;
	while (i < Math.sqrt(M))
	{
		if ((M%i)==0)
		{
				SM = SM +i;
				System.out.println(i+" ");
				SM = SM +(M/i);
				System.out.println(M/i+" ");
		}
		i = i+1;

	}
	System.out.println("somme = "+SM);


	SN = 1;
	i = 2;
	while (i < Math.sqrt(N))
	{
		if ((N%i)==0)
```

```
                {
                        SN = SN +i;
                        System.out.println(i+" ");
                        SN = SN +(N/i);
                        System.out.println(N/i+" ");
                }
                i = i+1;

        }
        System.out.println("somme = "+SN);




        }       // main
        //--------------------------------------------------------------------------//

}
```