Exercice 1

L'algorithme d'exponentiation modulaire rapide

1. (a) Calculer 2^{100} mod 5 avec scilab.
   (b) Calculer 2^{100} mod 5 « à la main »en utilisant que $2^{100} = (2^2)^{50}$
   (c) Sachant que $2^{100} = 1267650600228229401496703205376$ quel est le bon résultat ? Pourquoi l'autre résultat est il faux ?

2. Écrire une fonction scilab `E=binaire(e)` faible ».
   qui calcule l'écriture binaire de l'entier

   $$e = (e_n \dots e_1 e_0)_2 = e_n 2^n + \dots e_1 2 + e_0$$

   et la stocke dans un tableau

   $$E = \boxed{e_n}\boxed{e_{n-1}}\boxed{\dots}\boxed{e_1}\boxed{e_0}$$

   en suivant la méthode du « bit de poids

fonction $E =$ binaire(e)
$E =$ tableau pour stocker les e_i
tant que $e \neq 0$ **faire**
$x \equiv e \bmod 2$; (en fait $x = e_i$)
ajouter x dans le tableau E
$e = (e - x)/2$;
fin faire

3. En utilisant l'écriture binaire de e montrer que

   $$a^e = \prod_{\substack{i=0,\dots n \\ e_i \neq 0}} a^{2^i} \qquad \text{(produits des } a^{2^i} \text{ pour les } i \text{ tels que } e_i \neq 0). \tag{1}$$

4. Calculer, en fonction de a, les premiers termes de la suite et de i ci-dessous

   $$\begin{cases} u_0 & \equiv & a \bmod n \\ u_{i+1} & \equiv & u_i^2 \bmod n \end{cases} \tag{2}$$

   en déduire un moyen de calculer, par récurrence, les a^{2^i} mod n sans jamais manipuler de nombres supérieur à n^2 (donc valable tant que $n^2 < 10^{16}$).

5. À partir des questions précédente écrire une fonction Scilab `p=powermod(a,e,n)` calculant a^e mod n qui fonctionne dès que $a, e, n < 10^8$.

fonction $p =$ powermod(a, e, n)
$u = u_0 = a^{2^0}; p = 1$
tant que $e \neq 0$ **faire**
calculer e_i (de $e = (e_n \dots e_1 e_0)_2$)
si $e_i = 1$
alors ajouter u_i dans le produit p
fin
calculer $u_{i+1} \equiv u_i^2 \bmod n$
fin faire
p contient $a^e \equiv \prod_{\substack{i=0,\dots n \\ e_i \neq 0}} u_i \bmod n$

6. À l'aide de `powermod` vérifier que le nombre $2^{1193} + 1$ est divisible par 324497.

Exercice 2

Génération de nombres pseudo-aléatoires

La génération de nombres aléatoire est une question très importante en informatique. Les commandes disponibles dans différents langages sont en général basées sur ce qu'on appelle un générateur de congruences linéaires. l'idée est de trouver des coefficients a, b, n tels que la fonction $f(x) = ax + b \bmod n$ soit une bijection de $E = [0, n-1]$ dans lui même qui « mélange » les éléments de l'intervalle$[0, n-1]$. Le choix des coefficients a, b et n est assez compliqué :

- il faut prendre n assez grand pour que la période du générateur soit assez grande ,
- Les valeurs $a-1$ et b doivent au minimum être premières avec n

Le générateur de nombres `rand()` utilisé par Scilab est de ce type, ces coefficients ont été proposés par D. Knuth. Il renvoie une liste de nombres $(\frac{x_k}{n})_{k=1,\ldots} \subset [0,1[$ suivant les formules :

$$\begin{cases} x_{k+1} = f(x_k) = ax_k + b \bmod n \\ x_0 = la\ graine \\ rand() = \dfrac{x_k}{n} \in [0,1[\end{cases} \quad \text{avec} \quad \begin{array}{rcl} a & = & 843314861, \\ b & = & 453816693, \\ n & = & 2^{31} = 2147483648 \end{array}$$

La graine x_0 est initialisée par défaut à 0 au démarrage de scilab. On peut ce pendant réinitialiser la valeur de la graine de `rand` dans scilab avec la commande :

```
-->//valeur actuelle x0
-->rand("seed")
 ans  =
    1516320.
-->rand()
 ans  =
    0.6860406
```

```
-->//réinitialisation
-->rand("seed",0)


-->rand()
 ans  =
    0.2113249
```

```
-->//réinitialisation
-->rand("seed",1516320)


-->rand()
 ans  =
    0.6860406
```

1. Vérifier que la valeur renvoyée par `rand` au démarrage de scilab est bien 0.2113249 et quelle correspond bien à la valeur théorique annoncée.
2. Générer avec `rand` une matrice y de 10000 valeurs
   (a) tracer le graphe puis l'histogramme de ces valeurs :
   `x=[1:10000];y=rand(x);plot2d(x,y,5,axesflag=1); pause; histplot(10,y)`
   (b) calculer la moyenne et l'écart-type de y
   `y=rand(1,10000);mean(y),stdev(y)`
   et comparer avec les valeurs attendues pour une loi uniforme sur $[a, b]$:
   $$Y \sim \mathcal{U}([a,b]), \quad \mathbb{E}(Y) = \frac{a+b}{2} \qquad \sigma_Y = \frac{b-a}{\sqrt{12}}$$
   (c) calculer la moyenne du produit et le produit des moyennes pour deux séries $y1$ et $y2$ générées à partir de `rand` :
   `y1=rand(1,10000);y2=rand(1,10000);mean(y1.*y2),mean(y1)*mean(y2)`
   (d) justifier le caractère « pseudo-aléatoire » et « indépendant » des suites obtenues.

3. Écrire une fonction `L=Knuth(m,a,b,n)` qui renvoie une liste des m premières valeurs générées par la suite $(\frac{x_k}{n})$ définie par f avec les valeurs a, b, m en partant de la graine $x_0 = 0$. Retrouve-t-on bien les valeurs de `rand(1,m)` quand on reprend les paramètres choisis par Knuth ?
4. On fixe $n = 10^8$, choisir des coefficients a, b et tester votre générateur de nombres pseudo-aléatoire.
5. Écrire une fonction `M=myrand(p,n)` qui génère « aléatoirement » une matrice de réels $\in [0, 1[$ à partir de `rand` mais après avoir modifié la graine x_0 avec la formule [1] :

   $x_0 = (semaine+numero-1)\times(1+jour)\times(1+mois)\times(1+heure)\times(1+minute)\times(1+seconde)$

   en utilisant donc l'heure de lancement de la commande récupéré par `dt=getdate()` qui renvoie la date courante au format expliqué ci-dessous :
   - dt(1) : l'année (entier compris entre 0000 et 9999).
   - dt(2) : le mois (entier compris entre 01 et 12).
   - dt(3) : le numéro de semaine (entier compris entre 01 et 53).
   - dt(4) : le jour du calendrier Julien (entier compris entre 000 et 366).
   - dt(5) : *numéro* du jour de la semaine (entier compris entre 1 (dimanche) et 7).
   - dt(6) : le *jour* dans le mois (entier compris entre 01 et 31).
   - dt(7) : l'heure du jour (entier compris entre 00 et 23).
   - dt(8) : les minutes (entier compris entre 00 et 59).
   - dt(9) : les secondes (entier compris entre 00 et 59).
   - dt(10) : les millisecondes (entier compris entre 000 et 999).

```
-->dt=getdate()
 dt  =

    2010.    2.    5.    35.    5.    4.    13.    44.    26.    703.
```

6. Faire une fonction `L=random(m)` qui génère une liste de m entier à 8 chiffres (donc $< 10^8$) suivant la formule suivante :

fonction $x = \mathsf{random}(m)$
  $n = 10^8$
  tant que PGCD$(a, n) \neq 1$ **faire**
    dt = date courante (minutes, secondes, millisecondes)
    générer un $a \in [0, n]$ à partir de dt
  fin faire
  pour $i = 1$ **jusqu'à** m **faire**
    $x_i \equiv a^i \bmod n$
  fin faire

tester la fiabilité de ce générateur.

1. la formule donne un x_0 qui est compris entre 0 et $59 \times 32 \times 13 \times 25 \times 60 \times 60 \approx 2^{31}$.

Repères historiques, Donald Knuth : Donald Ervin Knuth (10 janvier 1938 à Milwaukee, Wisconsin) est un informaticien américain de renom et professeur émérite en informatique à l'Université de Stanford. Il est un des pionniers de l'algorithmique, et a fait de nombreuses contributions dans plusieurs branches de l'informatique théorique.

Knuth a reçu son bachelor's degree en mathématiques de l'Université Case Western Reserve. Sa première analyse d'algorithme remonte à l'été 1962. Knuth découvre un lien entre l'efficacité d'un algorithme de hachage et des mathématiques remontant à Ramanujan. Il obtient ensuite son doctorat en mathématiques au California Institute of Technology en 1963. En 1968, il devient membre de la faculté de l'Université de Stanford où un poste a été créé spécialement pour lui : Professor Emeritus of the Art of Computer Programming. Il recevra de nombreux prix internationaux (dont le prix Thuring en 1972) et est élu membre associé de l'Académie des sciences française en 1992 et membre de la Royal Society en 2003.

Knuth est connu comme l'auteur de l'ouvrage The Art of Computer Programming (couramment appelé TAOCP), une des références dans le domaine de l'informatique, pour ne pas dire la bible (un mot cher à Knuth...) des informaticiens. Ce livre a établi un domaine : l'analyse d'algorithmes qui consiste à se servir des mathématiques pour étudier les performances (en temps, mémoire,...) d'un algorithme sur l'ensemble de ses exécutions possibles.

Knuth est aussi le créateur du système de composition de documents TEXet du système de création de polices Metafont, et a inauguré le concept de programmation lettrée (literate programming). Knuth est une figure de l'informatique, connue pour son humour "geek" : il offre par exemple une prime de 2,56 dollars pour chaque faute typographique ou erreur découverte dans ses livres sous prétexte que "256 cents font un dollar hexadécimal" (pour les erreurs de son ouvrage 3 :16 Bible Texts Illuminated la prime est cependant de 3,16 dollars). Les numéros de version de TEXconvergent vers π, c'est-à-dire que les versions se suivent de la sorte : 3, 3.1, 3.14, etc, les numéros de version de Metafont convergent eux vers e. Il a également mis en garde les utilisateurs d'un de ses logiciels ainsi : " Faites attention aux bugs dans ce code ; je n'ai fait que le prouver, je ne l'ai pas essayé ". *source http ://fr.wikipedia.org/*