Outils Mathématiques pour l'Informatique
TP noté (Projet)

Licence 3 Informatique

Maxime Bros
Xlim, Université de Limoges

Novembre 2019

Instructions pour le TP noté :

- Ce projet est à réaliser **individuellement**, toutes similitudes suspectes entre deux rapports ou codes seront sanctionnées.
- Le langage à utiliser est Python 3 (comme en TP). La grande majorité des informations nécessaires sont dans le cours qui vous a été fourni. Tout appel à des modules ou fonctions non utilisés en TP ou mentionnés dans le sujet est interdit.
- Votre code doit s'exécuter sans erreur, c'est la base de tout travail en informatique. Un code qui ne s'exécute pas est noté sur la moitié des points. *Pensez à rendre vos programmes les plus ergonomiques possibles : affichages à l'écran, saisies clavier pour l'utilisateur, etc.*
- Le projet contiendra : vos fichiers de code (un ou deux par exercice maximum), un rapport au format pdf [1]. Le tout sera à mettre dans une archive (par exemple .zip) [2] portant votre nom.
- Trouvez le juste milieu pour le rapport : pas besoin de faire 20 pages, mais une demi page n'est pas suffisante. Décrivez ce que vous avez fait, ce que vous avez compris ou non, ce que font vos fonctions, comment utiliser vos programmes et enfin les réponses aux questions théoriques du projet.
- Le projet est à envoyer à l'adresse **maxime.bros@unilim.fr** pour le **12 Janvier 2020 à 23h au plus tard**, tout retard sera sanctionné [3]. Si vous avez rendu votre projet et que je n'ai pas accusé réception de ce dernier, n'hésitez pas à me relancer 24h après.
- Et enfin, pas de panique ! Le sujet semble long mais il n'y a pas énormement de choses à faire, il y a surtout des explications. Bonne chance à tous.

1. Si votre rapport contient des images, vous pourrez les ajouter dans l'archive, pour que je puisse les voir en plus grand.
2. C'est-à-dire que votre rendu de projet doit tenir en un unique fichier.
3. Malus de 2 points par tranche de 12h.

1 Collection de cadeaux

Dans certains restaurants, lors de l'achat d'un menu enfant, un jouet est offert. Imaginons que le jouet offert est choisi aléatoirement parmi n jouets différents, et ceci indépendamment des jouets offerts les fois précédentes.

Le but de cet exercice est de connaitre le nombre de menus à acheter pour espérer avoir la collection complète de tous les jouets différents. Évidemment cette valeur va dépendre de n. Par exemple si $n = 1$, il suffit de prendre un seul menu et la collection est déja complète !

Indication : si un événement arrive avec une probabilité p, il faut en moyenne réaliser $\frac{1}{p}$ expériences indépendantes pour voir cet événement arriver.

Question 1 : Calculer le nombre de menus qu'il faut acheter **en moyenne** pour compléter la collection avec $n = 2$ et $n = 3$. Détaillez votre raisonnement et vos calculs dans le rapport, il n'y a rien à programmer pour cette question.

Question 2 : Écrivez un programme qui demande à l'utilisateur de saisir la valeur de n et qui simule l'achat successif de menus enfants jusqu'à ce que la collection soit complète. Le programme affiche alors

"Il aura fallu ... menus pour que la collection soit complète."

Vous utiliserez pour cela la génération de nombres aléatoires vue en TP. Naturellement cette valeur changera à chaque exécution du programme puisque c'est le résultat d'une expérience aléatoire.

Si vous souhaitez utiliser des tableaux en Python 3, il suffit d'utiliser les quelques fonctions suivantes [4] :

```
tab=[] # creation d'un tableau vide
taille=len(tab) # donne la longueur du tableau
tab=tab+[a] # ajout d'un element 'a' dans tab
tab[i] # acces a la case i du tableau
[a]*n # tableau avec n fois la valeur a dedans
```

Question 3 : Modifiez le programme précédent : cette fois-ci l'utilisateur doit rentrer n (le nombre de jouets dans la collection) et m le nombre de réalisations de l'expérience. Vous afficherez alors le nombre de menus nécessaires pour compléter la collection **en moyenne** sur les m expériences.

4. Encore une fois, il est interdit d'utiliser des modules autres que ceux vus en TP ou décrits dans ce sujet.

Question 4 : Utilisez le programme précédent pour créer un nouveau programme qui demandera toujours n et m à l'utilisateur ; mais qui cette fois-ci générera un fichier contenant le nombre de menus nécessaires pour chaque valeur de 1 à n. C'est à dire que la première ligne contiendra 1, la deuxième la moyenne (sur les m tentatives) du nombre de tentatives nécessaires pour compléter une collection avec 2 jouets, etc, jusqu'à n jouets.

Dans votre rapport, expliquez la manière dont vous avez codé cela, éventuellement les optimisations apportées ; enfin, vous donnerez les temps approximatifs de calculs pour générer les fichiers (avec les valeurs (n, m) correspondantes).

Question 5 : Utilisez ce fichier de données pour tracer une courbe avec `gnuplot` comme en TP, du nombre de menus en fonction de n. Vous mettrez cette courbe (capture d'écran) dans le rapport et l'image originale dans l'archive.

Valeurs conseillées pour avoir un beau graphe : $n = 100, m = 1000$.

Question 6 : Cette courbe est très proche [5] du graphe de la fonction

$$f(n) = n \times (g(n) + 0.6)$$

où $g(n)$ est une fonction classique en mathématiques, à vous de trouver laquelle parmi les suivantes : fonction cube, fonction carré, exponentielle en base 2, logarithme népérien (ou naturel), racine carrée, sinus et cosinus.

Pour répondre à cette dernière question, votre programme doit créer deux fichiers distincts, l'un contenant les valeurs précédemment calculées et l'un contenant les valeurs de la fonction que vous aurez choisie pour tous les n successifs [6].

Vous joindrez alors le graphe contenant les deux courbes superposées dans votre rapport. Pour rappel, il faut pour cela d'abord taper la commande `gnuplot` dans un terminal, puis saisir la commande

```
plot "valeurs_menus.txt" w l, "valeurs_fonction.txt" w l
```

Pour cette question, vous avez le droit d'utiliser le module `math` de Python 3 car il contient toutes les fonctions ci-dessus (regarder la documentation pour connaître la syntaxe, par exemple `math.sqrt(n)` donne la racine carrée).

2 RSA

RSA (du nom de ses inventeurs : Rivest, Shamir, Adleman) est un système de chiffrement asymétrique inventé en 1977 et encore très largement utilisé de nos jours.

5. En fait, elle se rapproche très rapidement d'une autre courbe, en mathématiques on dit qu'elle converge vers elle.

6. Par exemple, si vous avez choisi racine carrée, $f(n) = n(\sqrt{n} + 0.6)$

Le terme “asymétrique” veut simplement dire que le chiffrement nécessite une clé différente de celle qui permet de déchiffrer. La clé pour le chiffrement est dite *publique* car elle peut-être partagée, la clé de déchiffrement est quant à elle *privée*.

Si je vous donne ma clé publique RSA [7], vous pouvez m'envoyer un message chiffré que je serai le seul à pouvoir déchiffrer en utilisant ma clé privée.

La clé publique est un couple (n, e), n est le modulo et e l'exposant ; la clé privée est simplement un entier d.

Pour chiffrer un message m (m doit être un entier entre 0 et $n-1$), il suffit de calculer le chiffré c de la façon suivante :

$$c \;=\; m^e \bmod n$$

Quand le destinataire du message reçoit le chiffré c, il retrouve le message initial en calculant

$$c^d \bmod n$$

La sécurité du chiffrement RSA repose sur la difficulé de retrouver d à partir de n. Si un attaquant dispose de la factorisation de n, c'est-à-dire l'expression de n comme un facteur de nombres premiers, il peut retrouver d. Néanmoins la factorisation d'entiers de grandes tailles est un problème difficile, c'est la raison pour laquelle les modulos utilisés en pratique pour RSA sont des nombres gigantesques.

Génération des clés RSA : supposons que $e = 65537$ [8], pour générer des clés RSA, il suffit de procéder ainsi :

1. Choisir deux “grands” nombres premiers distincts p et q et calculer $n = pq$
2. Il est très important de vérifier que e et $(p-1)(q-1)$ soient premiers entre eux, sinon on retourne à l'étape 1.
3. La clé privée d est simplement l'inverse de e modulo $(p-1)(q-1)$, c'est-à-dire $ed \equiv 1 \bmod (p-1)(q-1)$.

Vous voyez donc pourquoi il est important que n soit difficile à factoriser, car une fois que l'on connait p et q tels que $n = pq$, on peut calculer $(p-1)(q-1)$ avec une simple multiplication et d à partir de l'algorithme d'Euclide étendu vu en cours et en TP.

Le but de cet exercice est “d'attaquer” RSA en retrouvant un message à partir de son chiffré, sans connaitre la clé privée. La clé publique est

$$n = 1101559224973962301047827529028867976696486074114489216087217499217550543$$

$$e = 65537$$

7. Certaines personnes publient leur clé publique RSA sur leur site web.

8. En pratique, il existe des règles strictes à respecter pour le choix d'un “bon” exposant e, mais cela dépasse le cadre de ce projet. Vous verrez cela en Master Cryptographie.

et le chiffré est

$c = 436485656120232254281829097696606578007424709112490999398633583703939391$

Remarque : vous vous demandez sûrement pourquoi il est possible de casser RSA avec un n qui est pourtant très grand ? Tout simplement car cette valeur de n est en fait ridiculement petite comparée à des vrais modules RSA, qui valent environ 2^{1024} pour une sécurité raisonnable.

Question 1 : Essayer de factoriser n à l'aide de Sagemath, avec la commande de factorisation vue en TP. Est-ce que cela fonctionne ? Si non, pourquoi à votre avis ? Et si oui, en combien de temps ?

Question 2 : Essayer de factoriser n à l'aide de Sagemath, mais cette fois-ci avec la commande `qsieve(n)` [9]. Est-ce que cela fonctionne ? En combien de temps (en secondes) ? Joindre une capture d'écran de l'exécution de Sagemath sur laquelle apparaît ce temps.

Astuce : si vous écrivez `time qsieve(n)`, Sagemath vous donnera la factorisation et le temps du calcul. Sinon vous pouvez le calculer vous-même à l'aide du module `time` et de la commande `time.time()`.

Question 3 : Écrivez un programme qui demande à l'utilisateur de saisir 3 nombres : p, q et un chiffré c. Le programme affiche alors $n = pq$, la valeur de d (la clé privée RSA correspondant à la clé publique (n, e) avec $e = 65537$). Attention, si d n'existe pas, il faut afficher un message d'erreur.

Le programme affiche ensuite la valeur du message original m calculé à partir du chiffré c. Attention, pour élever le nombre c à la puissance d puis prendre le modulo, il ne faut surtout pas calculer d'abord c^d puis $c^d \bmod n$, car c^d est un nombre beaucoup trop grand. Vous pouvez pour cela utiliser la fonction `pow(a,b,n)` de Python 3.

Pour les élèves désireux d'avoir **un point bonus**, vous pouvez coder vous-même une fonction `square_and_multiply(a,b,n)` qui, exactement comme `pow`, calcule $a^b \bmod n$ à l'aide de la méthode décrite dans le fichier "square and multiply" de M. Clavier disponible sur Community.

Question 4 : Utilisez votre programme pour déchiffrer le chiffré c donné plus haut. Vous pourrez alors convertir le nombre obtenu en un message avec l'une des deux fonctions données ci-dessous, à vous de choisir laquelle. Vous donnerez alors le message secret que vous aurez réussi à retrouver [10].

9. Il s'agit d'un appel à l'algorithme "quadratic sieve" qui permet de factoriser de manière efficace des nombres de grandes tailles. Néanmoins il existe des algorithmes encore meilleurs.
10. Il s'agit d'une réplique culte d'un film d'espionnage français.

```
def str_to_int(message) :
    entier=bytes(message,"utf-8").hex()
    return int(entier,16)

def int_to_str(entier) :
    mess_hex=bytes.fromhex(hex(entier)[2:])
    return mess_hex.decode("utf-8")
```