Sujets Python - M2 280

Théo Lopès-Quintas

2022-2023

Consignes générales

Les sujets proposés ici sont à l'image des cours : la difficulté n'est pas dans les notions traitées, mais dans la capacité à produire un code de grande qualité. La notation suivra ce principe : un code qui répond parfaitement au sujet ne pourra avoir une note élevée si le code n'est pas simple, modulaire et lisible.
On portera un intérêt particulier aux noms des différentes variables et des fonctions. Les sujets étant volontairement simples, il est attendu pour avoir une meilleure note d'aller au delà du sujet : faites preuve d'initiative.

Il n'est attendu que le ou les scripts Python, mais si vous jugez nécessaire d'accompagner le code avec un fichier texte et/ou des captures d'écran de ce que produit le code c'est possible, mais il n'y a rien d'obligatoire.

1 Coder et décoder le code RSA

On souhaite crypter un message avec le code RSA et être capable de le décrypter en connaissant toute les informations, mais également sans connaître la clé secrète.

Prenons l'exemple de Bob qui veut échanger un message avec Alice en utilisant le code RSA. Commençons par la création de la clé publique et secrète.

1. Bob choisit deux nombres premiers p et q et calcule $n = p \times q$.
2. Alice calcule $\phi(n) = (p-1) \times (q-1)$ et choisit e tel que e et $\phi(n)$ soient premier entre eux.
3. Alice calcule d définit par : $d \times e \equiv 1[\phi(n)]$
4. Alice partage (n, e) la clé publique et d est appelé la clé secrète.

Ainsi, Alice donne à tout le monde la clé publique, mais n'échange qu'avec Bob la clé secrète. A partir de maintenant, Bob peut crypter son message et le transmettre à tout le monde. Seulement Alice sera cependant capable de le décrypter. Pour crypter et décrypter un message, on suit la procédure suivante.

1. Bob écrit un message et le transforme en un nombre m.
2. Bob calcule $c \equiv m^e[n]$ et transmet le message crypté c.
3. Alice calcule $c^d[n] = m$ puis traduit le message en texte.

Exemple 1. *On prend $p = 11$ et $q = 59$. On a donc que la clé publique est $(n, e) = (649, 3)$ et la clé secrète est $d = 387$.*
Bob souhaite transmettre le message m =theo. Pour ce faire, il utilise la fonction ord *qui renvoie un code pour chaque caractère. Ici, par exemple* $\mathsf{ord}(t) = 116$. *Puis, en utilisant le processus décrit au-dessus, on obtient la valeur 51.*

La grande faiblesse de la méthode RSA est quelle est très *simple* à décoder pour un attaquant, donc quelqu'un qui ne connait pas la clé secrète, mais uniquement la clé publique (n, e) et message crypté c. En effet, si l'on est capable de calculer la décomposition en nombre premier de n, on peut retrouver p et q et donc reconstruire d. La raison qui fait que le code est encore utilisé, c'est qu'en utilisant de très grand nombre premiers, il est très difficile de trouver cette décomposition en nombre premier. Par exemple, avec une implémentation naïve comme nous l'avons fait dans la séance 1 de la décomposition en facteur premier, il faut au moins plus de 20 minutes pour retrouver la décomposition en facteur premier de $n = 991 \times 997 = 988027$: j'ai arrêté le chronomètre à 20 minutes.

Il est donc demandé dans ce projet d'être capable de coder et de décoder n'importe quel message texte. On demande aussi d'être capable de décoder un message pour lequel on ne connait que la clé publique.

2 Conjecture de Goldbach

En 2018 est publié l'article *Goldbach's Function Approximation Using Deep Learning* par Avigail Stekel, Merav Chkroun et Amos Azaria. Le papier propose d'explorer les possibilités du Deep Learning pour aider les mathématiciens à trouver des contres exemples à certaines conjecture réputées difficiles.

La conjecture de Goldbach est que tout nombre pair strictement plus grand que 2 peut être représenté comme la somme de deux nombres premiers. Cependant, cette décomposition n'est pas unique. Ainsi, on définit la fonction de Goldbach $G(n)$ qui a chaque entier n associe le nombre de possibilités d'écriture de n comme la somme de deux nombres premiers. Ainsi, la conjecture de Goldbach peut se réécrire à l'aide de la fonction G comme l'impossibilité qu'il existe n pair et strictement supérieur à 2 tel que $G(n) = 0$.

Le papier présenté s'intéresse à approcher la fonction G avec du Deep Learning. On observe que les résultats atteint permettent d'améliorer les bornes classiques obtenues par les mathématiciens jusqu'ici.

Nous souhaiterions retrouver les résultats du papier. Pour cela, nous devons être capable, pour un nombre donné n de calculer l'ensemble des partitions possible. Egalement, nous souhaiterions avoir des fonctions *utilisateurs* qui permettent de visualiser l'ensemble des partitions, par exemple, de manières esthétiques.

Python étant un language plutôt lent, nous souhaiterions avoir un code un peu plus optimal en terme de vitesse d'exécution qu'une implémentation simple. Pour cela, n'hésitez pas à utiliser des astuces que nous aurions vu en cours.
Nous ne sommes pas intéressé par la partie Machine Learning / Deep Learning, seulement par la génération d'une table avec les différentes valeurs de la fonction G.

3 Jouer au Monopoly

On souhaite jouer au Monopoly, mais personne ne veut jouer avec nous parce qu'une partie est trop longue. Alors, nous nous proposons de produire une version numérique du jeu Monopoly.
Nous allons bien sur simplifier les règles. Nous souhaitons avoir deux types de joueurs possible :

- **Humain** : joueur physique auquel on demandera son avis pour acheter ou non une propriété, ainsi qu'investir ou non sur une propriété possédée en maison/hôtels.
- **Ordinateur** : joueur artificiel qui répondra au même question que le joueur humain mais de manière aléatoire. On ne cherche pas à avoir d'optimisation dans ce projet.

Le plateau de Monopoly correspond à 40 cases de plusieurs types :

- **Case départ** : que l'on numérote à 0. Chaque passage permet de faire gagner 200¤ au joueur.
- **Case impôt** : numéro 4 ou 38, où le joueur est obligé de payer la somme correspondante à la case.

- **Case prison** : numéro 10
- **Case *allez en prison*** : numéro 30. Nous ne traiterons pas ici de mécanisme de blocage pendant un certain nombre de tour, le joueur est simplement déplacé en prison.
- **Case *chance* ou *Caisse de communauté*** : numéro 2, 7, 17, 22, 33 ou 36. Nous ne traiterons pas ces cas, donc un joueur qui atteint ces cases n'aura aucune action.
- **Case *Parc gratuit*** : le joueur reçoit l'ensemble des paiements liés aux cases impôts, puis la valeur est réinitialisé à 0.
- **Case gare** : pour simplifier, on ne traitera pas cette mécanique. Ainsi un joueur qui atteint ces cases n'aura aucune action.

Chaque joueur commence avec 1600¤ et a un récapitulatif de sa situation financière avant chaque achat. Le joueur humain peut arrêter le jeu tous les 5 tours à sa demande, autrement le jeu s'arrête quand il ne reste plus qu'un seul joueur.

A l'aide d'un fichier qui recense les prix et coûts de chacune des propriétés, répliquer le jeu Monopoly extrêmement simplifié. Un visuel du plateau n'est pas attendu, mais la possibilité de pouvoir jouer à plusieurs joueurs (humain ou artificiel) l'est.

A Justification de la procédure RSA

Soit p et q premiers, on définit leurs produit $n = p \times q$ et on définit la fonction ϕ qu'on appelle indicatrice d'Euler.

L'indicatrice d'Euler se définit comme suit : pour un nombre donné a, elle renvoie le nombre de nombre premier avec a inférieur ou égal à a. Donc $\phi(8) = 4$ parce que seulement 1, 3, 5 et 7 sont premiers avec 8.
Ainsi, $\phi(p) = p-1$ pour p premier. La fonction a une propriété de multiplicativité pour a et b premiers entre eux : $\phi(ab) = \phi(a)\phi(b)$. Pour le prouver, il faut travailler avec le théorème des restes chinois et l'exploiter pour caractériser la correspondance entre $\{1, 2, \ldots, ab\}$ et le produit cartésien $\{1, 2, \ldots, a\} \times \{1, 2, \ldots, b\}$.

D'où la notation : $\phi(n) = (p-1)(q-1)$. On a choisit d comme l'inverse modulaire de e pour $\phi(n)$, autrement dit :

$$\exists k \in \mathbb{Z}, ed = 1 + k \times \phi(n)$$

On calcule le message m crypté c grâce à :

$$c = m^e[d]$$

Ainsi,

$$\begin{aligned} c^d[n] &= m^{de}[n] \\ &= m^{1+k\phi(n)}[n] \\ &= m \times \left(m^{\phi(n)}\right)^k [n] \end{aligned}$$

Or on sait par le théorème de Fermat généralisé que pour tout entier naturel n strictement positif et tout entier a premier avec n, on a :

$$a^{\phi(n)} \equiv 1[n]$$

D'où le résultat.