Robotique
CRC
Robotics

CRC

COMPÉTITION DE PROGRAMMATION

PROBLÈME PRÉLIMINAIRE 2

QUELQUES NOTES

- Les règles complètes sont dans la section 4 du livret des règlements
- Vous avez jusqu'au **Dimanche 28 janvier, 11:59 pm** pour remettre votre code
- N'hésitez pas à utiliser le forum de programmation sur le discord de la CRC pour poser vos questions et discuter des problèmes. Il est là pour ça!
- **On vous donne des fichiers modèles faciles à utiliser pour votre code et pour faire vos tests. Merci de les utiliser.**

UTILISATION DU FICHIER MODÈLE

- En gros, le fichier de test appelle la fonction associée avec en paramètre les informations du test et compare sa sortie avec ce qui est attendu pour vous permettre de voir si ça fonctionne. **Tout votre code (sauf fonctions additionnelles et classes que vous avez créé, qui devraient aller juste au-dessus) devrait être écrit dans la fonction prévue à cet effet.**
- Les points mis dans le document sont indicatifs de la difficulté de chaque section et du pointage pour leur réussite pour ce défi. Ce défi préliminaire va avoir une valeur globale de 2% du défi principal. Pour plus d'informations, aller voir la section 4 du livret des règlements.

STRUCTURE

Chaque problème contient une petite mise en situation comme celle-ci expliquant les fondements du problème et donnant les bases nécessaires pour résoudre celui-ci. Chaque problème préparatoire contient aussi la répartition des points pour le problème.

Spécification d'entrée et de sortie:

Dans cette section, on retrouve les caractéristiques des entrées qui peuvent être fournies au code en question ainsi que les critères attendus pour les sorties du programme.

Exemple d'entrée et de sortie:

Dans cette section se trouve un exemple d'entrée (parfois constitué lui-même de plusieurs sous-exemples) pour que vous puissiez tester votre code. L'exemple de sortie donne donc la réponse attendue pour cette entrée.

Explication de la première sortie:

Si le problème n'est toujours pas clair après la mise en situation, l'explication de la première sortie sert parfois à démêler le tout en expliquant comment la première entrée est traitée et en montrant le chemin menant à cette réponse.

Code Secret(ement compliqué)

Nous voulons établir un code secret pour transmettre des messages, mais comme ce document est public, ce sont vos capacités de programmation qui permettront de transmettre des messages cachés et maintenir le processus secret. Votre tâche consiste à établir 4 clés avec les 4 premières parties pour ensuite chiffrer le message dans la dernière.

Partie 1: Outille-moé comme du monde (30 points)

Vous avez beaucoup d'outils dans votre atelier et vous ne voulez pas les compter un par un. Pour compter plus rapidement vous faites des petits groupes d'outils. Des groupes de 5, de 7, de 23, des nombres premiers qui vous viennent à l'esprit.

Par contre, quand vous faites vos groupements, il reste quelques outils qui ne sont dans aucun groupement. Soyez heureux de savoir que c'est grâce à ces outils restants que vous saurez le nombre d'outils minimal en votre possession!

Vous aurez donc différents groupements et les restants pour vous aider à trouver le nombre minimum d'outils. Vous aurez besoin du nombre d'outils de chaque groupement sur le groupe d'outils que nous allons nommer g_1, g_2, g_3, … et aussi du nombre d'outils restants après un regroupement par paquet de g outils. Ces restants seront nommés r_1, r_2, r_3, …

La dernière série de nombre importants pour la formule sont le nombre minimal à multiplier tous les autres groupements g pour obtenir un de plus qu'un multiple du groupement. Nous allons appeler ces nombres m_1, m_2, m_3, … Voici la formule pour calculer m_1:

$$m_1 * (g_2 * g_3 * g_4 * \ldots) \equiv 1 \ (mod\ g_1)$$

Bref, comme tous les nombres d'outils par groupement g sont donnés, il faut trouver le plus petit m_i pour lequel tous les autres g multipliés ensemble donnent 1 de plus qu'un multiple de g_i.

Une fois le nombre m calculé pour chacun des groupements nous pouvons appliquer la formule pour découvrir un nombre possible d'outils x.

$$ProdG = g_1 * g_2 * g_3 * \ldots$$

$$x = (ProdG/g_1 * r_1 * m_1) + (ProdG/g_2 * r_2 * m_2) + (ProdG/g_3 * r_3 * m_3) + \ldots$$

Le x trouvé maintenant est un nombre d'outils possible, mais pas nécessairement le plus petit nombre. Le plus petit est le résultat du modulo avec la multiplication de tous les g.

$$min = x \bmod (ProdG)$$

Spécification d'entrée:

En entrée, vous recevrez un array contenant des tuples de nombre par groupement et restants associés, donc $[(g_1, r_1), (g_2, r_2), (g_3, r_3), (g_4, r_4), \dots]$

Spécification de sortie:

En sortie, vous devrez renvoyer une string correspondant au nombre d'outils minimal appelé min remplie avec des zéros au début afin de sortir une clé à 4 chiffres.

Exemples d'entrée:

```
[(3,2), (5,3), (7,2)]
[(21, 5), (2, 1), (19, 7), (5, 2)]
[(7,4), (11,2), (23,14), (3,2)]
```

Exemples de sortie:

```
0023
3617
5051
```

Explication de la première sortie:

On sait qu'on a 2 outils restants quand on fait des groupements de 3, on a 3 outils restants quand c'est des groupements de 5 et finalement 2 outils restants dans notre groupement par groupes de 7 outils.

Nous avons donc $g_1, g_2, g_3, r_1, r_2, r_3$ et avons maintenant besoin de trouver les m pour chacun des groupements. Trouvons le m du groupement par groupe de 3 avec la formule:

$$m_1 * (g_2 * g_3 * g_4 * \dots) \equiv 1 \ (mod\ g_1)$$

Nous souhaitons trouver le plus petit m positif pour lequel tous les autres g modulo g_1 donne 1.

Si on remplace les chiffres qu'on connait et qu'on essaie avec $m_1 = 1$

$$1 * (5 * 7)\ mod(3) \equiv 2$$

Le résultat du modulo qu'on souhaite avoir est 1 et nous obtenons 2. Le m_1 essayé n'est donc pas le bon. Nous essayons donc avec $m_1 = 2$

$$2 * (5 * 7)\ mod(3) \equiv 1$$

Cette fois-ci le résultat du modulo est 1 donc nous gardons $m_1 = 2$

Une fois le processus fait pour les 3 groupement on trouve $m_1 = 2, m_2 = 1, m_3 = 1$

On peut donc appliquer la formule pour obtenir x

$$x = (ProdG/g_1 * r_1 * m_1) + (ProdG/g_2 * r_2 * m_2) + (ProdG/g_3 * r_3 * m_3) + \dots$$

$$x = (105/3 * 2 * 2) + (105/5 * 3 * 1) + (105/7 * 2 * 1)$$

$$x = 140 + 63 + 30$$

$$x = 233$$

Avec notre x on peut maintenant terminer le problème et trouver le nombre minimum d'outils appelé min

$$min = x \, mod(ProdG)$$
$$min = 233 \, mod(105)$$
$$min = 23$$

Nous avons donc 23 outils au minimum dans le premier exemple!

Partie 2: Un petit pas pour l'homme, un grand… NOMBRE!?? (35 points)

Vous devrez ici vous familiariser avec une arithmétique littéralement extraterrestre: les nombres lunaires! Les nombres lunaires ont pour propriété de prendre les maximum et minimum de deux *digits* comme opérations principales. Ainsi, l'addition et la multiplication dans cette algèbre sont définies comme suit:

$$a + b = max(a, b)$$
$$a \times b = min(a, b)$$

Ainsi, $2 + 7 = 7$ mais $2 * 7 = 2$. Il faut cependant se méfier, comme les opérations se font directement entre *digits*, il faut vérifier que les opérations plus de taille fonctionnent toujours. Comme l'addition est déjà une opération qui fonctionne par paire de *digits*, tout va bien et, en comparant les colonnes,$956 + 379 = 979$, par exemple. Cependant, la multiplication est distributive entre les *digits* de deux nombres. Ainsi:

$$811 \times 81$$
$$= 111 + 8110$$
$$= 8111$$

Comme une multiplication normale écrite sous forme de colonne, il faut en réalité multiplier 811 par 1 d'abord, puis par 80. En additionnant les *digits* respectifs, on obtient donc finalement bel et bien 8111, et non 811 ou 11, comme on aurait pu s'y attendre naïvement à partir de notre définition de multiplication. Il faudra donc faire attention lors de multiplications.

L'avantage principal de cette algèbre farfelue est qu'elle ne contient pas de restant à ramener au *digit* suivant lorsque la multiplication sous forme de colonne est utilisée. Par exemple, 4*7 donne 8 à l'unité mais nécessiterait d'additionner un 2 supplémentaire à la dizaine en algèbre normale. Ici, en arithmétique lunaire, les *digits* de la réponse peuvent théoriquement être calculés indépendamment l'un de l'autre.

Spécification d'entrée:

En entrée, vous recevrez une string composée de deux nombres entiers positifs séparés simplement par une opération de l'arithmétique lunaire (simplement + ou *).

Spécification de sortie:

En sortie, vous devrez renvoyer le résultat du calcul demandé selon les lois bizarres de l'arithmétique lunaire.

Exemples d'entrée:

```
811*81
345+4567
57*557
```

Exemples de sortie:

```
8111
4567
5557
```

Explication de la première sortie:

Voir explication dans la description du problème.

Partie 3: Où est Charlie!? (15 points)

On sait tous comment trouver Charlie habituellement, mais il vient de se déguiser en nombre! Il est allé se cacher parmi une longue séquence de chiffres et il faudra le trouver pour qu'il réponde enfin de ses actes.

Heureusement, notre informateur sait en quel nombre il est camouflé, il suffira de retrouver ce nombre dans un « tableau ». Pour un nombre et une séquence de chiffres donnés, il suffira d'indiquer la position du dernier chiffre du nombre trouvé (le dernier chiffre de Charlie) pour obtenir la clé.

Spécification d'entrée:

En entrée, vous recevrez une longue string contenant un peu moins de 10 000 chiffres séparés aléatoirement par d'autres caractères, ainsi qu'un nombre entier positif à 9 chiffres (Charlie est long comme 9 chiffres!) qu'il faudra retrouver dans la séquence.

Spécification de sortie:

En sortie, vous devrez renvoyer un nombre entier positif correspondant à la position du **dernier chiffre** de Charlie. En cas de plusieurs occurrences (chose fort peu probable), il ne faudra considérer que la première occurrence.

Exemples d'entrée:

```
Le début et la fin de la séquence ont été coupés pour rendre plus
lisible et aucun marquage particulier n'est dans les vrais tests.

grande_string:
…0324k39460492499123456789565423105781086579872507889213941606121x031
5198246031562446000217102554251380531600291 0…

sequence_a_trouver:
123456789
```

Exemples de sortie:

```
position: 1579
```

Explication de la première sortie:

Dans cet exemple, dans la version complète de la grande string, le dernier caractère de la séquence à trouver est à la position 1579 en commençant le compte avec le premier caractère étant le caractère 0.

Partie 4: Basiquement (40 points)

Le but de cette partie est d'établir un protocole pour convertir n'importe quel nombre en clé à 4 *digits* seulement. La première étape de transformation est de changer le nombre en base 10 à un nombre dans une autre base, puis d'exprimer les *digits* en base 10 pour former un nouveau nombre concaténé et le transformer encore plus. Il faudra faire cette étape 5 fois. Par exemple, prenons la base 29 et le nombre 982:

$(982)_{10} \Rightarrow (14P)_{29} \Rightarrow 1425$ (P correspond à 25)

$(1425)_{10} \Rightarrow (1K4)_{29} \Rightarrow 1204$

$(1204)_{10} \Rightarrow (1CF)_{29} \Rightarrow 11215$

$(11215)_{10} \Rightarrow (D9L)_{29} \Rightarrow 13921$

$(13921)_{10} \Rightarrow (GG1)_{29} \Rightarrow 16161$

La première étape encoderait donc 982 à 16161. Ensuite, la deuxième étape consiste à classer les chiffres du nombre résultant selon le résultat d'un modulo 4. Ainsi, suivant le même exemple:

$$1 \equiv 1(mod4)$$

$$6 \equiv 2(mod4)$$

Finalement, pour trouver la clé, il faut additionner tous les *digits* d'un même groupe de congruence modulaire plusieurs fois jusqu'à ce qu'il ne reste qu'un seul chiffre entre 0 et 9. Donc, avec 16161:

0 (groupe 0 mod 4)

1+1+1=3 (groupe 1 mod 4)

6+6=12, 1+2=3 (groupe 2 mod 4)

0 (groupe 3 mod 4)

Concaténation finale de ces 4 chiffres donne la clé 0330 que nous allons retourner sous forme de string pour conserver le formatage.

Spécification d'entrée:

En entrée, vous recevrez deux nombres entiers positifs, le premier correspondant à la base de conversion (entre 2 et 99, vous n'avez pas besoin d'exprimer le nombre converti dans la base avec des lettres, ce n'était que pour démontrer une étape intermédiaire), et le deuxième correspondant au nombre d'entrée à encoder.

Spécification de sortie:

En sortie, vous devrez renvoyer un string représentant un nombre entier positif correspondant à la clé à 4 chiffres

Exemples d’entrée:

```
29 982
23 1053
65 812
```

Exemples de sortie:

```
0330
8400
0707
```

Explication de la première sortie:

Voir explication du début du problème.

Partie 5: Notions avancées de Cryptographie (25 points)

Tous les nombres obtenus précédemment ne sont pas inutiles! Votre partenaire d'affaires et vous avez établi un moyen élaboré de communication confidentielle à l'aide du présent problème, mais il faut une dernière couche de sécurité pour couronner le tout. Vous devrez chiffrer le message selon le protocole de chiffrement RSA. D'abord, il faut différencier les nombres obtenus qui serviront de parties de clé du nombre qui sera le message. **Le chiffrement RSA nécessite deux nombres premiers *p* et *q*.** À l'aide de ces deux nombres premiers il est possible de construire deux autres nombres fortement utiles:

$$n = p * q$$

$$\varphi(n) = (p - 1) * (q - 1)$$

Cela suffit pour les deux nombres premiers. Par la suite, il vous faudra un nombre qui **est premier avec (et inférieur à)** $\varphi(n)$ (C't'arrangé avec le gars des vues). Il faut rappeler que deux nombres premiers entre eux sont deux nombres qui ne partagent aucun diviseur excepté 1. Par exemple, 4 est premier avec 7, mais 4 n'est pas simplement premier. Ce nombre est appelé *l'exposant de chiffrement* et sera dénoté par la variable *e*. **Le dernier nombre est donc par élimination le message à transmettre *M*. Le message sera codé de la façon suivante:**

$$C \equiv M^e \ (mod\ n)$$

où *C* est le message chiffré et *mod* est l'opération *modulo*. Il ne reste qu'à vérifier qu'il est possible de déchiffrer le message avec les informations obtenues aussi des parties 1 à 4. Il faudra donc vérifier le calcul en trouvant *l'exposant de déchiffrement* écrit *d*. *d* est l'inverse *modulo* $\varphi(n)$ de *e*:

$$d * e \equiv 1 \ (mod\ \varphi(n))$$

Il y a plusieurs méthodes existantes pour calculer cet exposant de déchiffrement. La première étant que, par les propriétés des *modulo*, on doit trouver une paire d'entiers *u* et *v* satisfaisant:

$$e * u + \varphi(n) * v = 1$$

Une telle paire d'entiers peut être trouvée notamment par itérations ou avec l'algorithme d'Euclide étendu. À vous de trouver votre méthodologie de choix! De cette paire, *u* correspond à l'inverse modulaire de *e* et devra être ramené à sa plus simple forme pour correspondre à *d*. Par exemple, si nous avons *e=4* et $\varphi(n)$=7, *u* peut être -5, 2, 9, 51, etc. Cependant, seulement 2 se situe dans l'intervalle "principal" de 0 à 7 et ainsi, seulement 2 sera accepté comme *d*. Pour déchiffrer le message, il suffit de faire l'opération suivante:

$$M \equiv C^d \ (mod\ n)$$

Dans le doute de pouvoir évaluer de si grands nombres, il est utile de rappeler que ce sera toujours possible d'évaluer le tout par ordinateur en raison de la propriété des modulos suivante:

si $a \equiv b \ (mod\ n)$, alors $a^k \equiv b^k \ (mod\ n)$

Il est donc possible d'évaluer l'expression exponentielle « progressivement ». **Vos ordinateurs devraient être capable d'évaluer *C* sans utiliser ceci, mais pas d'évaluer *M* pour vérifier la validité de *d* sans si c'est quelque chose que vous souhaitez (*C* et *d* étant de l'ordre de *M* et *e* au carré, plus ou moins)**.

Précision sur RSA: ici, ce n'est pas exactement comment l'algorithme de chiffrement RSA est utilisé habituellement, mais cela donne une bonne introduction sur le sujet. *(e, n)* est la clé publique et *(d, n)* est la clé privée. Normalement, celui qui envoie un message ne connaît pas les nombres premiers *p* et *q*, et il n'est donc pas aussi simple de déterminer *d*, ce qui constituerait un problème de sécurité. En théorie, seul le "destinataire" connaît ces deux nombres, qui sont généralement énormes. L'algorithme se base sur la grand défi qu'est le problème de factorisation en nombres premiers afin d'établir une clé publique qui en révèle très peu sur la clé privée. **Si vous voulez plus d'informations sur certains aspects de ce problème, le chiffrement RSA, l'algorithme d'Euclide et l'algèbre modulaire sont tous des sujets assez bien répandus.**

Spécification d'entrée:

En entrée, vous recevrez 4 strings représentant les nombres entiers distincts dans l'ordre suivant: *p*, *q*, *e*, *M*. (Le même ordre que celui donné par les parties!)

Spécification de sortie:

En sortie, vous devrez d'abord donner un tuple composé d'un nombre entier correspondant au message chiffré *C*, suivi d'un nombre entier correspondant à l'exposant de déchiffrement *d*.

Exemples d'entrée:

```
p:0023, q:8111, e:1579, M:0330
p:3617, q:4567, e:3517, M:8400
p:5051, q:5557, e:3833, M:0707
```

Exemples de sortie:

```
(168442, 127459)
(13038683, 8379733)
(26895292, 16016297)
```

Explication de la première sortie:

Avec *p* et *q*, on trouve d'abord *n=186553* et $\varphi(n)$*=178420*. Le calcul du message chiffré donne par la suite *C=168442* et l'inversion modulaire donne *d=127459*.