Concours National Commun (CNC), Maroc

Session 2023, filière TSI

Informatique

Durée : 2 heures

Ce que traite le sujet

- Exercice : intervalles des entiers
- Partie I : base de données et SQL
- Partie II : calcul numérique
- Partie III : problème

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/cnc/tsi/2023/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

المملكة المغربية

وزارة التعليم العالي

والبحث العلمي والابتكار

ⵜⴰⴳⵍⴷⵉⵜ ⵏ ⵍⵎⵖⵔⵉⴱ

ⵜⴰⵎⴰⵡⴰⵙⵜ ⵏ ⵜⵣⵔⴼⵜ ⴰⵏⴰⵎⵓⵔ ⴰⵙⵙⵏⵜⵉⵔ

ⴰⵎⴰⵡⴰⵙ ⴰⵏⴰⵎⵓⵔ ⴷ ⵜⵙⵙⵏⵜⵉⵔⵜ



Royaume du Maroc

Ministère de l'Enseignement Supérieur,

de la Recherche Scientifique et de l'Innovation





CNC 2023

Concours National Commun

d'Admission dans les Établissements de Formation d'Ingénieurs et Établissements Assimilés

Épreuve de **Informatique**

Filière : **TSI**

Durée **2 heures**

Cette épreuve comporte **12 pages au format A4**, en plus de la page de garde

L'usage de la calculatrice est interdit.

Les candidats sont informés que la précision des raisonnements algorithmiques ainsi que le soin apporté à la rédaction et à la présentation des copies seront des éléments pris en compte dans la notation. Il convient en particulier de rappeler avec précision les références des questions abordées. Si, au cours de l'épreuve, un candidat repère ce qui peut lui sembler être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre

Remarques générales :

- ✓ Cette épreuve est composée d'un exercice et de trois parties tous indépendants ;
- ✓ Toutes les instructions et les fonctions demandées seront écrites en Python ;
- ✓ Les questions non traitées peuvent être admises pour aborder les questions ultérieures ;
- ✓ Toute fonction peut être décomposée, si nécessaire, en plusieurs fonctions.

***NB** : Le candidat doit impérativement commencer par traiter toutes les questions de l'exercice ci-dessous, et écrire les réponses dans les premières pages du cahier de réponses.*

Exercice : (4 points)

Intervalles des entiers

Q.1- Écrire la fonction ***affiche (a, b)*** qui reçoit en paramètres deux entiers **a** et **b** tels que **a≤b**, et qui affiche les entiers compris entre ***a*** et ***b***.

Exemples : ***affiche (2, 9)*** affiche les nombres **2, 3, 4, 5, 6, 7, 8** et **9**.

Q.2- Écrire la fonction ***appartient (x, a, b)*** qui reçoit en paramètres trois entiers **x**, **a** et **b** tels que **a≤b**, et qui retourne **True** si **x** appartient à l'intervalle des entiers $[\![a\ ,\ b]\!]$, sinon, la fonction retourne **False**.

Exemples :

- ***appartient (5, 2, 9)*** retourne **True**
- ***appartient (15, 2, 9)*** retourne **False**.

Q.3- Écrire la fonction ***somme (a, b)*** qui reçoit en paramètres deux entiers **a** et **b** tels que **a≤b**, et qui retourne la somme des entiers appartenant à l'intervalle des entiers $[\![a\ ,\ b]\!]$.

Exemple : ***somme (5, 12)*** retourne **68 = 5+6+7+8+10+11+12**

Q.4- Déterminer la complexité de la fonction ***somme (a, b)***, avec justification.

Q.5- Écrire la fonction ***produit (a, b)*** qui reçoit en paramètres deux entiers **a** et **b** tels que **a≤b**, et qui retourne le produit des entiers appartenant à l'intervalle des entiers $[\![a\ ,\ b]\!]$.

Exemple : **produit (2, 7)** retourne **5040 = 2*3*4*5*6*7**

Q.6- Écrire la fonction ***liste_elements (a, b)*** qui reçoit en paramètres deux entiers **a** et **b** tels que **a≤b**, et qui retourne une liste contenant les éléments de l'intervalle des entiers $[\![a\ ,\ b]\!]$.

Exemple : ***liste_elements (2, 9)*** retourne la liste **[2, 3, 4, 5, 6, 7, 8, 9]**

Q.7- Écrire la fonction ***intervalle (L)*** qui reçoit en paramètre une liste ***L*** de nombres entiers triés dans l'ordre croissant. La fonction retourne **True** si ***L*** représente un intervalle, sinon, la fonction retourne **False**.

Exemples :

- ***intervalle ([4, 5, 6, 7, 8, 9])*** retourne **True**
- ***intervalle ([4, 7, 8, 9, 10])*** retourne **False**

Partie I : Base de données et langage SQL

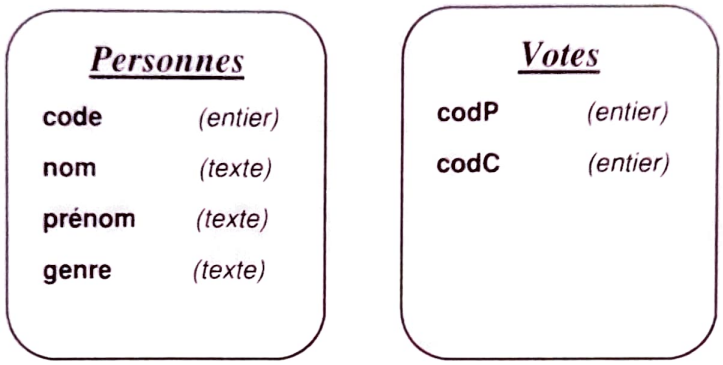
Élections

Dans un groupe de plusieurs personnes, on propose d'organiser des élections selon les règles suivantes :

- ✓ Une personne peut voter pour n'importe quelle personne, y compris elle-même ;
- ✓ Une personne a le droit de voter deux fois, mais pas pour la même personne.

NB : Les candidats aux élections sont des personnes pour qui on a voté.

Pour gérer les résultats de ces élections, on propose d'utilise une base de données composée de **2** tables : '**Personnes**' et '**Votes**'.



Structure de la table 'Personnes'

La table '***Personnes***' contient des informations sur les personnes qui ont le droit de voter, et les personnes qui sont candidats aux élections. Cette table est composée de **quatre** champs :

- Le champ ***code*** contient un entier unique permettant d'identifier chaque personne ;
- Le champ ***nom*** contient les noms des personnes ;
- Le champ ***prénom*** contient les prénoms des personnes ;
- Le champ ***genre*** contient ***'F'*** pour féminin ou ***'M'*** pour masculin.

Exemples :

code	*nom*	*prénom*	*genre*
362	LAHDOUD	AMAL	F
616	DIBBOU	MOHAMED	M
404	SAHAB	ZAYNAB	F
95	GUIDER	YASSER	M
282	BOUDANI	ALI	M
81	FASKA	RYM	F
77	IKIDOU	SALIM	M
...	...	...	...

Structure de la table 'Votes'

La table '***Votes***' contient les votes exprimés par les personnes. Cette table est composée de **deux** champs :

- Le champ ***codP*** contient les codes des personnes ayant voté ;
- Le champ ***codC*** contient les codes des personnes candidats aux élections.

Exemples :

codP	*codC*
362	95
362	282
616	95
616	81
95	95
404	282
282	95
77	282
...	...

Par exemple, la personne ayant le code ***362*** a voté pour la personne ayant le code ***95***.

Q. 1 – Écrire la requête SQL, qui permet d'ajouter dans la table ***Personnes*** les enregistrements suivants :

code	*nom*	*prénom*	*genre*
123	GUISSI	MARYAM	F
138	FORA	AHMED	M

Q. 2 – Écrire en algèbre relationnelle, la requête qui donne les codes, les noms et les prénoms des personnes qui ont voté pour le candidat ayant le code ***394***.

Q. 3 – Écrire la requête SQL, qui donne les codes, les noms et les prénoms des personnes qui ont voté pour le candidat ayant le code ***394***, triés dans l'ordre croissant des codes.

Q. 4 – Écrire la requête SQL, qui donne les codes, les noms, les prénoms et les genres des candidats dont le nom contient au moins **5** caractères et le deuxième caractère du nom est '**A**'. Le résultat doit être trié dans l'ordre alphabétique des noms et prénoms.

Q. 5 – Écrire la requête SQL, qui donne les codes, les noms et les prénoms des personnes du genre féminin, qui n'ont pas voté.

Q. 6 – Écrire la requête SQL, qui donne les codes, les noms, les prénoms et les genres des candidats, et le compte des votes pour chaque candidat, tels que le compte des votes est supérieur à ***1000***. Le résultat doit être trié dans l'ordre décroissant des comptes de votes.

Q. 7 – Écrire la requête SQL, qui donne le plus grand, le plus petit et la moyenne des comptes de votes des candidats.

Partie II : Calcul numérique

Interpolation polynômiale

Dans cette partie, on suppose que les modules **numpy** et **matplotlib.pyplot** sont importés

```
import numpy as np
import matplotlib.pyplot as plt
```

En analyse numérique, l'**interpolation polynômiale** est une technique d'interpolation d'un ensemble de données ou d'une fonction par un polynôme.

Étant donnés ***n*** points dans le plan ***(x0, y0)***, ***(x1, y1)***, ..., ***(xi, yi)***, ..., ***(xn-1, yn-1)***, avec les ***xi*** distincts deux à deux, on cherche le polynôme ***A*** de degré au plus ***n-1***, et qui passe par ces points.

Supposons que le polynôme d'interpolation est le suivant :

$$A(x) = a_0 + a_1x^1 + a_2x^2 + \cdots + a_{n-1}x^{n-1} \qquad (1)$$

Afin que le polynôme ***A(x)*** passe par l'ensemble des points (x_i, y_i) à interpoler, il doit vérifier : $A(x_i) = y_i$

Ainsi, on obtient un système d'équations linéaires ***M*A=Y***, dont l'écriture matricielle est la suivante :

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-2} & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-2} & x_1^{n-1} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-2} & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_{n-2} \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Pour construire le polynôme ***A(x)***, il faudra donc résoudre ce système afin d'obtenir les valeurs des coefficients a_i du polynôme ***A(x)***.

Q.1- Écrire la fonction ***decoupe (P)*** qui reçoit en paramètre une liste ***P*** de tuples qui représentent les points à interpoler. La fonction construit et retourne deux vecteurs ***X*** et ***Y*** qui contiennent respectivement les abscisses et les ordonnées des points de ***P***.

Exemple :

P = [(-1, 2) , (0, 1) , (5, 4) , (1, 3) , (-2, 8) , (-7, 3) , (3, -3)]

La fonction **decoupe (P)** retourne les vecteurs :

X = [-1 , 0 , 5, 1 , -2 , -7, 3] et ***Y*** = [2 , 1 , 4, 3 , 8 , 3 , -3]

Q.2- Écrire la fonction ***matrice (X)*** qui reçoit en paramètre la liste ***X*** des abscisses x_i des point à interpoler, et qui construit et retourne la matrice ***M*** suivante :

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-2} & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-2} & x_1^{n-1} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-2} & x_{n-1}^{n-1} \end{pmatrix}$$

Exemple :

X = [-1 , 0 , 5, 1 , -2 , -7, 3]

La fonction **matrice (X)** retourne la matrice ***M*** suivante :

```
[ [  1.   -1.    1.    -1.     1.     -1.      1.  ]
  [  1.    0.    0.     0.     0.      0.      0.  ]
  [  1.    5.   25.   125.   625.   3125.  156250.  ]
  [  1.    1.    1.     1.     1.      1.      1.  ]
  [  1.   -2.    4.    -8.    16.    -32.     64.  ]
  [  1.   -7.   49.  -343.  2401.  -16807.  117649.  ]
  [  1.    3.    9.    27.    81.    243.    729.  ] ]
```

Le module **numpy** contient un sous module **linalg**. Ce sous module contient la méthode **solve** qui permet de résoudre le système ***M*A=Y***.

La méthode **solve** reçoit en paramètres la matrice ***M*** et le vecteur ***Y***, et elle retourne le vecteur ***A***, solution du système, et qui contient les coefficients du polynôme ***A(x)***.

Q.3- Écrire la fonction ***polynome (P)*** qui reçoit en paramètre la liste ***P*** de tuples qui représentent les points à interpoler. La fonction calcule et retourne le vecteur ***A*** qui contient les coefficients du polynôme interpolateur passant par ces points.

Exemple :

X = [-1 , 0 , 5, 1 , -2 , -7, 3] et ***Y*** = [2 , 1 , 4, 3 , 8 , 3 , -3]

La fonction **polynome (X, Y)** retourne le vecteur ***A*** suivant :

[1. , 1.18720238 , 1.643125 , -0.71178571 , -0.14696429 , 0.02458333 , 0.00383929]

Q.4- Écrire la fonction ***image (A, x)*** qui reçoit en paramètres un réel ***x***, et le vecteur ***A*** contenant les coefficients du polynôme interpolateur ***A(x)***. La fonction retourne l'image de ***x*** par le polynôme interpolateur ***A***.

Exemple :

A **= [1. , 1.18720238 , 1.643125 , -0.71178571 , -0.14696429 , 0.02458333 , 0.00383929]**

La fonction **valeur (*A*, *4*)** retourne la valeur suivante :

$1.*4^0 + 1.18720238*4^1 + 1.643125*4^2 + -0.71178571*4^3 + \ldots + 0.0038929*4^6 = -10.23928571428571$

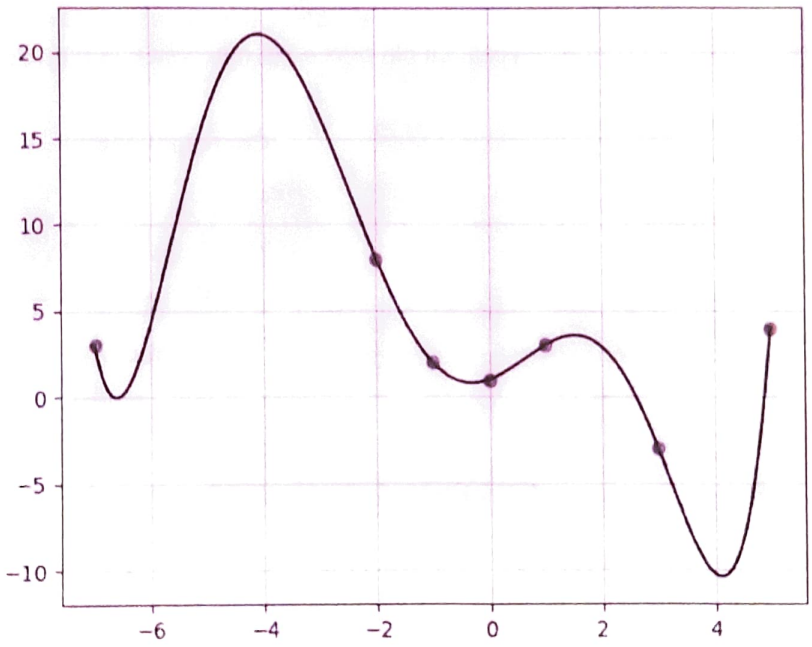
Q.5- Écrire la fonction ***courbe (P)*** qui reçoit en paramètre la liste ***P*** de tuples qui représentent les points à interpoler. La fonction trace la représentation graphique du polynôme interpolateur ***A(x)*** qui passe par tous les points de ***P***.

Le nombre de points générés dans la courbe est : **500**

Exemple :

P **= [(-1, 2) , (0, 1) , (5, 4) , (1, 3) , (-2, 8) , (-7, 3) , (3, -3)]**

Après l'appel de la fonction **courbe (P)**, on obtient la représentation du polynôme interpolateur qui passe par tous les points de ***P***.



.....

.....

.....

Partie III : Problème

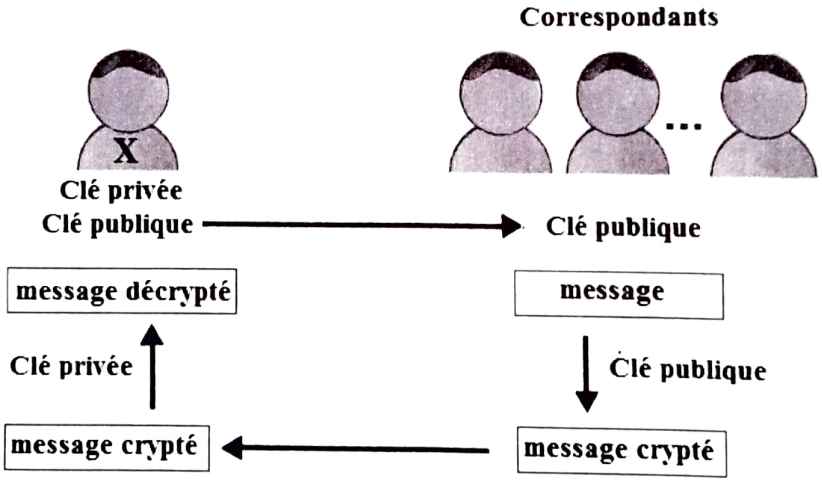
Nombres premiers et cryptologie

Le chiffrement RSA

Avec le développement d'internet, le besoin de transmettre des informations confidentielles de façon sécurisée, par exemple des numéros de carte bancaire, est en effet devenu primordial... C'est là notamment qu'intervient l'algorithme RSA, un algorithme de cryptographie basé sur une propriété simple des nombres premiers.

Nommé par les initiales de ses trois inventeurs : **R**onald Rivest, Adi **S**hamir et Leonard **A**dleman, le **chiffrement RSA** est un algorithme de cryptographie, très utilisé dans le commerce électronique, et plus généralement pour échanger des données confidentielles sur Internet. On estime que plus de 300 millions de programmes installés peuvent utiliser le RSA, les transactions sécurisées via internet par exemple l'emploient pour la plupart. Internet fait un usage systématique du RSA pour assurer la confidentialité du courrier électronique et authentifier les utilisateurs.

Le chiffrement RSA est asymétrique : il utilise une paire de clés (deux nombres entiers) composée d'une ***clé publique*** pour chiffrer et d'une ***clé privée*** pour déchiffrer des données confidentielles. Les deux clés sont créées par une personne, par convention nommée ***X***, qui souhaite que lui soient envoyées des données confidentielles. La personne ***X*** rend la clé publique accessible. Cette clé est utilisée par ses correspondants pour chiffrer les données qui lui sont envoyées. La clé privée est réservée à la personne ***X***, et lui permet de déchiffrer ces données.



Une condition indispensable est qu'il soit impossible de déchiffrer à l'aide de la seule clé publique, en particulier de reconstituer la clé privée à partir de la clé publique, c'est-à-dire que les moyens de calcul disponibles, les méthodes connues au moment de l'échange et le temps que le secret doit être conservé ne le permettent pas. Le couple (clé privée, clé publique) est engendré par un procédé vraiment aléatoire qui, même s'il est connu, ne permet pas de reconstituer la clé privée.

L'algorithme RSA utilise les congruences sur les entiers et le petit théorème de Fermat, pour obtenir des fonctions à sens unique. Tous les calculs se font modulo un nombre entier ***n*** qui est le produit de deux nombres premiers. Le petit théorème de Fermat joue un rôle important dans la conception du chiffrement. Les messages clairs et chiffrés sont des entiers inferieurs a l'entier ***n***. Les opérations de chiffrement et de déchiffrement consistent à élever le message à une certaine puissance modulo ***n*** (c'est l'opération d'exponentiation modulaire).

L'objectif de cette partie est d'implémenter l'algorithme du **chiffrement RSA**, en langage python.

Rappel :

En Python, l'opérateur ***modulo*** est représenté par le symbole %.

a%b renvoie le reste de la division entière de **a** par **b**.

Exemples :

- **17%5** renvoie le reste **2** (17=3*5+2)
- **12%6** renvoie le reste **0** (12=2*6+0)
- **12%15** renvoie le reste **12** (12 = 0*15+12)

Calcul du PGCD par la méthode d'Euclide

On considère l'algorithme d'Euclide, qui permet de calculer le plus grand commun diviseur (PGCD) de deux entiers strictement positifs ***a*** et ***b*** :

> ***Entrée*** *: Deux entiers strictement positifs* ***a*** *et* ***b***
> ***r*** ← ***a*** *modulo* ***b***
> *Tant que* ***r*** ≠ ***0*** *faire*
> ***a*** ← ***b***
> ***b*** ← ***r***
> ***r*** ← ***a*** *modulo* ***b***
> *Fin Tant que*
> ***Sortie : b***

Exemple : Calcul du PGCD de **145** et **60**

a	***b***	***r = a modulo b***
145	60	25
60	25	10
25	10	5
10	5	0

Ainsi, le PGCD de **145** et **60** est **5**.

Q.1- Écrire la fonction ***pgcd (a, b)*** qui reçoit en paramètres deux entiers strictement positifs ***a*** et ***b***, et qui retourne le plus grand commun diviseur de ***a*** et ***b***, en utilisant l'algorithme d'Euclide.

Nombres premiers entre eux

Deux nombres entiers strictement positifs ***a*** *et* ***b*** *sont premiers entre eux si leur* ***PGCD*** *est égal à* ***1***.

Q.2- Écrire la fonction ***premiers (a, b)*** qui reçoit en paramètres deux entiers strictement positifs ***a*** et ***b***, et qui retourne **True** si ***a*** et ***b*** sont premiers entre eux, sinon, la fonction retourne **False**.

Exemples :

- **premiers (63, 20)** retourne **True**
- **premiers (115, 75)** retourne **False**

Équation et coefficients de Bézout

L'équation de Bézout est une équation à deux variables entières ***x*** et ***y*** de la forme :

$$a * x + b * y = c$$

Cette équation admet des solutions entières uniquement si le nombre **c** est divisible par le PGCD de ***a*** et ***b***. Ainsi, l'équation $a * x + b * y = 1$ admet des solutions quand ***a*** et ***b*** sont premiers entre eux.

Pour trouver une solution de l'équation de Bézout $a * x + b * y = 1$, on utilise l'algorithme d'Euclide.

Par exemple, on considère les deux nombres premiers entre eux **120** et **23**, et on propose de trouver une solution de l'équation de Bézout : $120x + 23y = 1$

Pour cela, on commence par rassembler dans une liste, les valeurs intermédiaires de ***a*** et ***b***, dans l'algorithme d'Euclide :

a	***b***	***r = a modulo b***
120	23	5
23	5	3
5	3	2
3	2	1
2	1	0

Les valeurs intermédiaires de ***a*** et ***b*** sont : **(120, 23)**, **(23, 5)**, **(5, 3)**, **(3, 2)** et **(2, 1)**

Q.3- Écrire la fonction ***liste_valeurs (a, b)*** qui reçoit en paramètres deux entiers strictement positifs ***a*** et ***b*** premiers entre eux. La fonction retourne une liste de tuples contenant les valeurs intermédiaires de ***a*** et ***b***, dans l'algorithme d'Euclide.

Exemple :

liste_valeurs **(120, 23)** retourne la liste **[(120, 23), (23, 5), (5, 3), (3, 2), (2, 1)]**

Ensuite, on utilise la liste des valeurs intermédiaires dans le sens inverse, comme illustré dans le tableau suivant :

	u , v = v, u-(a/b)*v		
(a, b)	***u=1***	***v=0***	***a*u + b*v = 1***
(2, 1)	**0**	1 - (2/1)*0 =**1**	2***0** + 1***1** = 1
(3, 2)	**1**	0 - (3/2)*1 = **-1**	3***1** + 2*(**-1**) = 1
(5, 3)	**-1**	1 - (5/3)*(-1) = **2**	5*(**-1**) + 3***2** = 1
(23, 5)	**2**	(-1) - (23/5)*2 = **-9**	23***2** + 5*(**-9**) = 1
(120, 23)	**-9**	2 - (120/23)*(-9) = **47**	120*(**-9**) + 23***47** = 1

Ainsi, le tuple **(-9, 47)** est une solution de l'équation de Bézout $120x + 23y = 1$.

*(en remplaçant **x** par **-9** et **y** par **47**, on obtient : **120*(-9) + 23*47 = 1**)*

Q.4- Écrire la fonction ***coeff_bezout (a, b)*** qui reçoit en paramètres deux entiers strictement positifs ***a*** et ***b*** premiers entre eux. La fonction retourne un tuple ***(u, v)*** solution de l'équation de Bézout : $a * x + b * y = 1$

Exemple : ***coeff_bezout*** **(120, 23)** retourne le tuple **(-9, 47)**

Calcul de la clé publique et la clé privée du chiffrement RSA

Algorithme de création de la clé publique et la clé privée :

Entrée : **p** et **q** deux entiers positifs distincts premiers *(suffisamment grands)*
n ← **p*q**
m ← **(p-1)*(q-1)**
Pour **k = m-1** à **2** pas **-1** faire
    Si **k** et **m** sont premiers entre eux Alors
        (d, v) ← une solution de l'équation de Bézout **k*x + m*y=1**
        Si **2<d<m** Alors
            Arrêt de la boucle Pour
        Fin Si
    Fin Si
Fin Pour
Sortie : **[(n, k) , (n, d)]**

NB : le tuple **(n, k)** représente la clé publique, et le tuple **(n, d)** représente la clé privée.

Pour tester si un entier strictement positif est premier ou non, on utilisera la propriété suivante :

*Un entier strictement positif **x** est **premier** s'il ne possède pas de diviseur entre **2** et* $\sqrt{x}$

Q.5- Écrire la fonction ***est_premier (x)*** qui reçoit en paramètre un entier strictement positif ***x***, et qui retourne **True** si ***x*** est premier, sinon, la fonction retourne **False**.

NB : Le nombre ***1*** n'est pas premier.

Exemples :

- **est_premier (1)** retourne **False**
- **est_premier (2)** retourne **True**
- **est_premier (15)** retourne **False**
- **est_premier (23)** retourne **True**

Q.6- Écrire la fonction ***keys (p, q)*** qui reçoit en paramètres deux entiers positifs distincts ***p*** et ***q*** premiers. En utilisant l'algorithme de création des deux clés privée et publique ci-dessus, la fonction retourne une liste contenant deux tuples : le premier tuple représente la clé publique, le deuxième tuple représente la clé privée.

Exemple :

keys (719, 127) retourne la liste des deux clés : **[(91313, 90463) , (91313, 36187)]**

- Le tuple **(91313, 90463)** représente la clé publique
- Le tuple **(91313, 36187)** représente la clé privée.

Calcul de puissance

L'algorithme de chiffrement **RSA** est basé sur le calcul des puissances de nombres avec de grands exposants. La première façon de calculer une puissance d'un nombre est de multiplier ce nombre par lui-même plusieurs fois. Cependant, il existe des méthodes bien plus efficaces.

Q.7- Écrire la fonction ***binaire (n)*** qui reçoit en paramètre un entier ***n*** strictement positif. La fonction retourne une liste ***B*** qui contient la conversion binaire de ***n***. *(Le premier bit à gauche est le bit du poids le plus fort)*

Exemple :

binaire (90463) retourne la liste : **B = [1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1]**

L'algorithme suivant est un algorithme de complexité logarithmique, qui calcule x^n, en utilisant la liste ***B*** contenant la conversion binaire de ***n*** :

Entrée : un entier positif ***x***, et une liste ***B*** qui contient la conversion binaire de ***n***

$p \leftarrow 1$

Pour ***i=0*** à ***taille(B)-1*** faire

    $p \leftarrow p^2$

    Si l'élément $B_i = 1$ Alors

        $p \leftarrow p*x$

    Fin Si

Fin Pour

Sortie : *p* *(**p** contient la valeur de x^n)*

Q.8- Écrire la fonction ***puissance (x, B)*** qui reçoit en paramètres un entier positif ***x*** et une liste ***B*** contenant la conversion binaire d'un entier ***n*** strictement positif. La fonction calcule et retourne la valeur de x^n, en utilisant l'algorithme ci-dessus.

Exemple :

puissance (67, [1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1]) retourne la valeur de 67^{90463}

Préparation du message pour le chiffrement

Le message initial est représenté par une liste contenant les codes des caractères du message. Par exemple, les codes des caractères du message "**Concours CNC TSI-2023**" sont :

C	o	n	c	o	u	r	s		C	N	C		T	S	I	-	2	0	2	3
67	111	110	99	111	117	114	115	32	67	78	67	32	84	83	73	45	50	48	50	51

Donc, le message **'Concours CNC TSI-2023'** est représenté par la liste ***T*** suivante :

T = [67, 111, 110, 99, 111, 117, 114, 115, 32, 67, 78, 67, 32, 84, 83, 73, 45, 50, 48, 50, 51]

Chiffrement RSA du message

Le tuple ***(n, k)*** étant la clé publique. On considère un élément ***x*** de la liste ***T*** et qui représente le code d'un caractère. Le chiffrement RSA de ***x*** consiste à calculer la valeur suivante :

$$c = x^k \ modulo \ n$$

Ainsi, le nombre ***c*** est le résultat du chiffrement RSA de ***x***.

Q.9- Écrire la fonction ***chiffrement_RSA (T, n, k)*** qui reçoit en paramètres la liste ***T*** des codes des caractères d'un message, et ***n***, ***k*** deux entiers qui représentent la clé publique. La fonction effectue le chiffrement RSA des éléments de ***T***, et elle retourne le résultat dans une nouvelle liste ***C***.

Exemple :

T = [67, 111, 110, 99, 111, 117, 114, 115, 32, 67, 78, 67, 32, 84, 83, 73, 45, 50, 48, 50, 51]

(n, k) = (91313, 90463) est la clé publique.

chiffrement_RSA (*T*, n, k) retourne la liste : **C = [22908, 27430, 48853, 70334, 27430, 74925, 73376, 79642, 75192, 22908, 1109, 22908, 75192, 70148, 49512, 34644, 50604, 48425, 24557, 48425, 32970]**

Déchiffrement RSA

On considère la liste ***C***, résultat du chiffrement **RSA** des éléments de la liste ***T***. Chaque élément ***c*** de ***C*** est calculé à l'aide d'un élément ***x*** de ***T*** et de la clé publique ***(n, k)***, en utilisant la formule suivante :

$$c = x^k \ modulo \ n$$

Le tuple ***(n, d)*** est la clé privée. À l'aide des coefficients de Bézout et du petit théorème de Fermat, on peut démontrer que :

$$x = c^d \ modulo \ n$$

Donc, réciproquement, on peut calculer la valeur de ***x*** à l'aide de **c**, **d**, et **n**.

Q.10- Écrire la fonction ***dechiffrement_RSA (C, n, d)*** qui reçoit en paramètres une liste ***C*** résultat du chiffrement RSA d'une liste ***T***, et ***n***, ***d*** deux entiers qui représentent la clé privée. La fonction effectue le déchiffrement des éléments de ***C***, et elle retourne la liste ***T*** initiale.

Exemple :

C = [22908, 27430, 48853, 70334, 27430, 74925, 73376, 79642, 75192, 22908, 1109, 22908, 75192, 70148, 49512, 34644, 50604, 48425, 24557, 48425, 32970]

(n, d) = (91313, 36187) est la clé privée.

dechiffrement_RSA (C, n, d) retourne la liste initiale :

T = [67, 111, 110, 99, 111, 117, 114, 115, 32, 67, 78, 67, 32, 84, 83, 73, 45, 50, 48, 50, 51]

********* FIN *********

Travailler les chapitres de ce sujet

Prepa Help suit le programme de TSI chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Architecture de l'ordinateur et représentation des nombres
- Bases de Python : expressions, types, instructions, fonctions
- Récursivité
- Complexité des algorithmes itératifs et récursifs

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/cnc/tsi