Épreuve d'Informatique – Session 2023 – Filière MP

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)

Les nombres amis

***x** étant un entier strictement positif. Un entier **i** est un **diviseur strict** de **x**, si **i** divise **x** et **i** différent de **x**.*

Exemple : les diviseurs stricts du nombre entier **8** sont **1**, **2** et **4**.

Q.1- Écrire la fonction ***divStrict (x)*** qui reçoit en paramètre un entier strictement positif **x**, et qui affiche les diviseurs stricts de **x**.

Exemple : ***divStrict (8)*** affiche les nombres **1, 2** et **4**

Q.2- Déterminer la complexité de la fonction ***divStrict (x)***, avec justification.

Q.3- Écrire la fonction ***somDivStrict (x)*** qui reçoit en paramètre un entier strictement positif **x**, et qui retourne la somme des diviseurs stricts de **x**.

Exemple : ***somDivStrict (8)*** retourne le nombre **7= 1+2+4**

*Deux entiers strictement positifs **x** et **y** sont **amis**, si :*

- ➢ ***x** est égal à la somme des diviseurs stricts de **y** ;*
- ➢ *et **y** est égal à la somme des diviseurs stricts de **x**.*

Q.4- Écrire la fonction ***amis (x, y)*** qui reçoit en paramètres deux entiers strictement positifs ***x*** et ***y*** et qui retourne **True** si ***x*** et ***y*** sont amis, sinon, la fonction retourne **False**.

Exemple : ***amis (284, 220)*** retourne **True**

Q.5- Écrire la fonction ***liste_amis (n)*** qui reçoit en paramètre un entiers ***n*** strictement positif. La fonction retourne une liste ***L*** qui contient les tuples de nombres amis ***(i , j)*** tels que ***0<i≤j≤n***.

Exemple : ***liste_amis (300)*** retourne la liste **[(6, 6), (28, 28), (220, 284)]**

Q.6- Déterminer la complexité de la fonction ***liste_amis (n)***, avec justification.

Partie I : Base de données et langage SQL

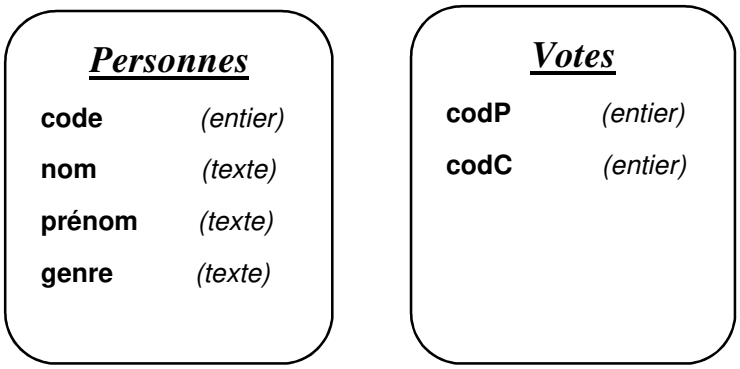
Élections

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 soi-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 masculin, qui n'ont pas voté.

Q. 6 – On suppose qu'un candidat a réussi d'obtenir plus que **50%** du compte total des votes exprimés.

Écrire la requête SQL qui donne le code, le nom, le prénom et le genre de ce candidat.

Q. 7 – On suppose qu'aucun candidat n'a réussi d'obtenir plus que **50%** du compte total des votes exprimés.

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 de Newton

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 newtonienne**, du nom d'Isaac Newton, est une méthode d'interpolation polynomiale permettant d'obtenir le polynôme de Lagrange comme combinaison linéaire de polynômes de la « base newtonienne ». Cette méthode ne diffère de l'interpolation lagrangienne que par la façon dont le polynôme est calculé, le polynôme d'interpolation qui en résulte est le même. Pour cette raison on parle aussi plutôt de la forme de Newton du polynôme de Lagrange.

Étant donnés n points dans le plan (x_0, y_0), (x_1, y_1), …, (x_i, y_i), …, (x_{n-1}, y_{n-1}), avec les abscisses x_i distincts deux à deux. L'interpolation polynomiale dans une base de Newton est une combinaison linéaire de polynômes appartenant à cette base :

$$N(x) = \sum_{j=0}^{n-1} a_j n_j(x)$$

Les polynômes de Newton sont définis de la manière suivante :

$$n_j(x) = \prod_{i=0}^{j-1}(x - x_i) \qquad avec, en\ particulier : n_0 = 1$$

Une méthode de calcul direct des a_j serait de résoudre le système triangulaire inférieur d'équations linéaires suivant :

$$\sum_{j=0}^{i} a_j n_j(x_i) = y_i \qquad pour\ i = 0, 1, 2, \dots, n-1$$

Qui s'écrit :

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & x_1 - x_0 & 0 & \cdots & 0 \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & \cdots & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & x_{n-1} - x_0 & \cdots & \cdots & \prod_{j=0}^{n-2}(x_{n-1} - x_j) \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_{n-2} \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

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

Les éléments x_i dans ***X*** et y_i dans ***Y*** sont respectivement l'abscisse et l'ordonnée du point p_i dans ***P***.

Exemple :

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

La fonction **decoupe (P)** retourne les deux vecteurs ***X*** et ***Y*** :

***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 retourne la matrice triangulaire inférieure ***M*** suivante :

$$\begin{pmatrix} 1 & 0 & 0 & & 0 \\ 1 & x_1 - x_0 & 0 & & 0 \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & x_{n-1} - x_0 & \dots & \dots & \prod_{j=0}^{n-2}(x_{n-1} - x_j) \end{pmatrix}$$

Exemple :

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

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

```
[ [  1.   0.   0.    0.     0.     0.     0.  ]
  [  1.   1.   0.    0.     0.     0.     0.  ]
  [  1.   6.   3.    0.     0.     0.     0.  ]
  [  1.   2.   2.   -8.     0.     0.     0.  ]
  [  1.  -1.   2.  -14.    42.     0.     0.  ]
  [  1.  -6.  42.  -504.  1680.  20160.    0.  ]
  [  1.   4.  12.  -24.   -48.   -240.  -2400. ] ]
```

Q.3- Écrire la fonction ***solution (M, Y)*** qui reçoit en paramètres la matrice triangulaire inférieure ***M*** (résultat de la fonction précédente), et le vecteur ***Y*** contenant les ordonnées des points à interpoler. La fonction renvoie le vecteur ***A*** solution du système triangulaire inférieur ***M*A=Y***.

NB : Pour calculer les éléments A_i du vecteur ***A***, on utilise la formule suivante :

$$A_i = \frac{1}{M_{i,i}} * (Y_i - \sum_{j=0}^{i-1} (M_{i,j} * A_j))$$

Exemple :

La fonction **solution (M, Y)** retourne le vecteur ***A*** suivant :

[2. , -1. , 0.26666667 , -0.30833333 , 0.00357143 , 0.00922619 , 0.00383929]

Pour évaluer le polynôme d'interpolation en un point d'abscisse x, on utilise la formule de Newton suivante :

$$\sum_{i=0}^{n-1} (A_i * (\prod_{j=0}^{i-1} x - x_j))$$

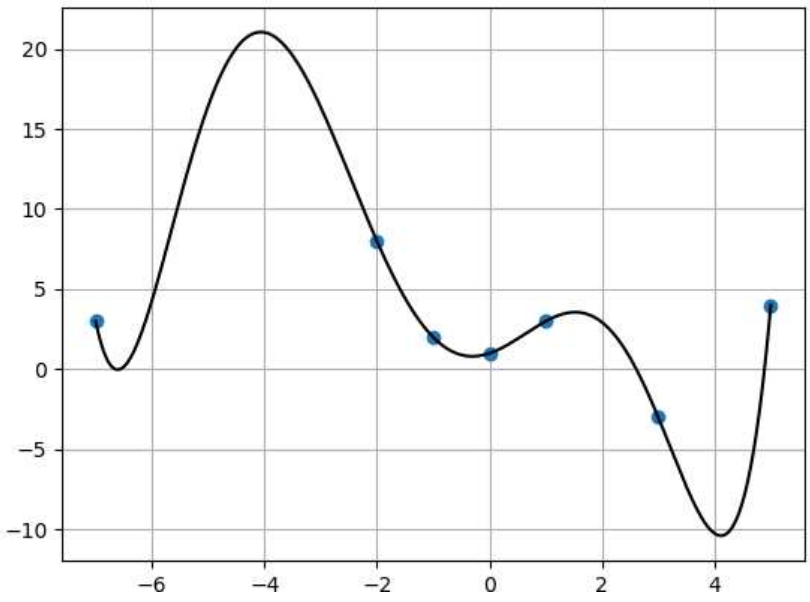
Q.4- Écrire la fonction ***valeur (x, A, X)*** qui reçoit en paramètres un réel x; le vecteur ***A*** solution du système triangulaire inferieure et le vecteur ***X*** des abscisses des points à interpoler. La fonction calcule et retourne la valeur du polynôme d'interpolation en x.

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 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 graphique du polynôme interpolateur qui passe par tous les points de ***P***.



Partie III : Problème

Fonctions de hachage

La fonction 'SHA-256'

Une **fonction de hachage**, de l'anglais *hash function* (*hash* : pagaille, désordre, recouper et mélanger par analogie avec la cuisine) est une fonction particulière qui, à partir d'une donnée fournie en entrée, calcule une valeur de sortie servant à identifier rapidement la donnée initiale, au même titre qu'une signature pour identifier une personne. La donnée d'entrée de ces fonctions est souvent appelée ***message*** ; la valeur de sortie est souvent appelée, **empreinte numérique**, ou **haché**.

Une fonction de hachage idéale possède les propriétés suivantes :

- la fonction est déterministe, c'est-à-dire qu'un même message aura toujours la même valeur de hachage ;
- il est impossible, pour une valeur de hachage donnée, de construire un message ayant cette valeur ;
- il est impossible de trouver deux messages différents ayant la même valeur de hachage (résistance aux collisions) ;
- la modification dans le message modifie considérablement la valeur de hachage.

Les fonctions de hachage sont utilisées en informatique et en cryptographie. Une application importante du hachage cryptographique est la vérification de l'intégrité d'un fichier ou d'un message. Par exemple, la modification d'un fichier lors d'une transmission (ou toute autre activité) peut être prouvée en comparant la valeur de hachage du fichier avant et après la transmission.

Une autre application du hachage cryptographique est la vérification de mot de passe. Le stockage de tous les mots de passe utilisateur en clair peut entraîner une violation de sécurité massive si le fichier de mots de passe est compromis. Une façon de réduire ce danger est de stocker seulement la valeur de hachage de chaque mot de passe. Pour authentifier un usager, le mot de passe fourni par l'utilisateur est haché et comparé avec la valeur de hachage stockée. Avec cette approche, les mots de passe perdus ne peuvent pas être récupérés s'ils sont oubliés ; ils doivent être remplacés par de nouveaux mots de passe.

Les fonctions de hachage, y en a plusieurs : MD4, MD5, SHA-1, SHA-224, SHA-256, SHA-384, …. Dans la suite de cette partie, nous nous intéresseront à la fonction de hachage **SHA-256**.

En 2002, la **fonction de hachage SHA-256** devient un standard fédéral de traitement de l'information. À partir d'un texte de taille maximale $\mathbf{2^{64}}$ bits (c.à.d. 2 Péta-octets), cette fonction produit un haché de **256** bits, représenté par **64** caractères hexadécimaux.

Exemples :

- Le haché du message **'Concours CNC 2023'** est :
  'ff33dfa67e2c471aa85a83c7a4632955594a946f38ac4a5904873ad5c64f9894'
- Le haché du message **'Concours CNC MP-2023'** est :
  '0f3503b5ac59d7160ed4c579d7548100c326c47a8673f36031e27d42e099e3a3'

L'objectif de cette partie est d'implémenter l'**algorithme du hachage SHA-256**, en langage python.

A- Manipulation des bits

Q.1- Écrire la fonction ***NON (X)*** qui reçoit en paramètre une chaîne de caractères ***X*** de taille **32**, contenant une représentation binaire. La fonction retourne la chaîne de caractères ***Z*** de taille **32**. Les éléments z_i de la chaîne ***Z*** sont calculés en utilisant l'algorithme suivant :

x_i est un élément d'indice ***i*** dans ***X*** :

- **Si x_i = '0' alors z_i = '1'**
- **Si x_i = '1' alors z_i = '0'**

Exemple : NON ('01000011010011100100001100100000') retourne la chaîne de caractères :
'10111100101100011011110011011111'

Q.2- Écrire la fonction ***ET (X, Y)*** qui reçoit en paramètres deux chaînes de caractères ***X*** et ***Y*** de même taille **32**, contenant respectivement deux représentations binaires. La fonction retourne la chaîne de caractères ***Z*** de taille **32**. Les éléments z_i de la chaîne ***Z*** sont calculés en utilisant l'algorithme suivant :

x_i et y_i sont respectivement deux éléments de ***X*** et ***Y***, de même indice ***i*** :

- **Si x_i = '1' et y_i = '1' alors z_i = '1'**
- **Si x_i = '0' ou y_i = '0' alors z_i = '0'**

Exemple : ET ('01000011010011100100001100100000', '00110010001100000011001000110011')
retourne la chaîne : **'00000001000000000000000001000100000'**

Q.3- Écrire la fonction ***XOR (X, Y)*** qui reçoit en paramètres deux chaînes de caractères ***X*** et ***Y*** de même taille **32**, contenant respectivement deux représentations binaires. La fonction retourne la chaîne de caractères ***Z*** de taille **32**. Les éléments z_i de la chaîne ***Z*** sont calculés en utilisant l'algorithme suivant :

x_i et y_i sont respectivement deux éléments de ***X*** et ***Y***, de même indice ***i*** :

- **Si $x_i = y_i$ alors z_i = '1'**
- **Si $x_i \neq y_i$ alors z_i = '0'**

Exemple : XOR ('01000011010011100100001100100000', '00110010001100000011001000110011')
retourne la chaîne : **'01110001011111110011100010010011'**

Q.4- Écrire la fonction ***decal_droite (X, p)*** qui reçoit en paramètres une chaîne de caractères ***X*** de taille **32**, contenant une représentation binaire, et un entier ***p*** tel que $1 \leq p < 32$. La fonction décale les bits de ***X*** de ***p*** positions vers la droite : *Supprimer les **p** derniers bits de **X**, et ajouter **p** fois le bit **'0'** au début de **X**.*

Exemple : decal_droite ('0100001101001110010000011**00100111', 8)** retourne la chaîne de caractères :
'0000000001000011010011100100001 1**'**

Q.5- Écrire la fonction ***rotation_droite (X, p)*** qui reçoit en paramètres une chaîne de caractères ***X*** de taille **32**, contenant une représentation binaire, et un entier ***p***, tel que $1 \leq p < 32$. La fonction effectue une rotation des bits de ***X*** de ***p*** positions vers la droite : *Déplacer les derniers **p** bits de **X** au début de **X**.*

Exemple : rotation_droite ('0100001101001110010**001100100111', 12)** retourne la chaîne de caractères :
'0011001001110100001101001110010 0**'**

Q.6- Écrire la fonction ***addition (X, Y)*** qui reçoit en paramètres deux chaînes de caractères ***X*** et ***Y*** de même taille ***32***, contenant respectivement deux représentations binaires. La fonction retourne la chaîne de caractères de taille ***32***, résultat du calcul en modulo 2^{32}, de l'addition binaire de ***X*** et ***Y***.

L'addition binaire utilise le principe suivant :

- **0 + 0 = 0**
- **0 + 1 = 1**
- **1 + 0 = 1**
- **1 + 1 = 10** *(on pose 0 et on retient 1)*
- **1+1+1 = 11** *(on pose 1 et on retient 1)*

Exemple : addition ('11111111111111111111111111111010' , '00000000000000000000000000011100') retourne la chaîne de caractères : **'00000000000000000000000000010110'**

B- Initialisation des variables de hachage

L'algorithme SHA-256 utilise **8** variables globales : ***h0***, ***h1***, ***h2***, ***h3***, ***h4***, ***h5***, ***h6***, ***h7*** et ***h8***. Ces variables sont initialisées respectivement par les chaînes de caractères contenant les premiers **32** bits des représentations binaires, des parties fractionnaires des racines carrées des **8** premiers nombres premiers : **2**, **3**, **5**, **7**, **11**, **13**, **17** et **19**.

Exemple :

La racine carrée de **2** est **1.4142135623730951**. Sa partie fractionnaire est **0.4142135623730951**
La représentation binaire de cette partie fractionnaire est : 0.01101010000010011110011001100111111…
La variable ***h0*** est initialisée par les premiers **32** bits : **'01101010000010011110011001100111'**

L'algorithme SHA-256 utilise aussi une variable globale ***K*** initialisée par une liste composée de **64** constantes. La liste ***K*** est initialisée respectivement par les chaînes de caractères contenant les premiers **32** bits des représentations binaires, des parties fractionnaires des racines cubiques des **64** premiers nombres premiers : **3**, **5**, **7**, **11**, **13**, **17**, **19** …. et **311**.

On suppose que ***L*** est une variable globale initialisée par les **64** premiers nombres premiers :

***L* = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, … , 269, 271, 277, 281, 283, 293, 307, 311]**

NB : Pour calculer $\sqrt[n]{x}$ (la racine nième d'un entier **x**), on utilise la syntaxe suivante : **x**(1/n)**

Q.7- Écrire la fonction ***bin_fract (x, n)*** qui reçoit en paramètres un nombre premier ***x*** et entier ***n*>1**. La fonction retourne la chaîne de caractères contenant les premiers **32** bits de la représentation binaires, de la partie fractionnaire de $\sqrt[n]{x}$.

Exemples :

- **bin_fract (7, 2)** retourne la chaîne de caractères : **'10100101010011111111010100111010'**
- **bin_fract (7, 3)** retourne la chaîne de caractères : **'11101001101101011101101110100101'**

Q.8- Écrire la fonction ***liste_init (t, n)*** qui reçoit en paramètres un entier strictement positif ***t ≤ 64*** et un entier ***n>1***. La fonction retourne une liste composée des chaînes de caractères contenant les premiers **32** bits des représentations binaires, des parties fractionnaires des racines $n^{ièmes}$ des ***t*** premiers nombres premiers.

Exemple : liste_init (4, 2) retourne la liste :

['01101010000010011110011001100111' , **'10111011011001111010111010000101'** ,

'00111100011011101111001101110010' , **'10100101010011111111010100111010']**

Q.9- Écrire les instructions permettant d'initialiser les variables globales de hachage *: **h0**, **h1**, **h2**, **h3**, **h4**, **h5**, **h6**, **h7**, **h8*** et ***K***.

C- Préparation du message

Le message qui sera haché est représenté par une liste contenant les codes des caractères du message.

Par exemple, les codes des caractères du message "**Concours CNC MP-2023**" sont :

C	**o**	**n**	**c**	**o**	**u**	**r**	**s**		**C**	**N**	**C**		**M**	**P**	**-**	**2**	**0**	**2**	**3**
67	*111*	*110*	*99*	*111*	*117*	*114*	*115*	*32*	*67*	*78*	*67*	*32*	*77*	*80*	*45*	*50*	*48*	*50*	*51*

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

T = [67, 111, 110, 99, 111, 117, 114, 115, 32, 67, 78, 67, 32, 77, 80, 45, 50, 48, 50, 51]

NB : On suppose que les éléments de la liste ***T*** sont tous des entiers positifs inférieurs strictement à **256**.

La préparation du message consiste à transformer la liste des codes ***T*** en une chaîne de caractères ***B*** dont la taille est un multiple de **512**, et qui contient les conversions binaires des éléments de ***T***, ainsi que la conversion binaire de la taille de ***T***. La chaîne ***B*** sera ensuite découpée dans une liste de listes.

Q.10- Écrire la fonction ***binaire (x, t)*** qui reçoit en paramètres deux entier strictement positifs ***x*** et ***t*** tels que $0 \leq x < 2^t$. La fonction retourne une chaîne de caractères de taille ***t***, contenant la conversion binaire de ***x***.

Exemple : binaire (67, 8) retourne la chaîne de caractères de taille **8 : '01000011'**

Q.11- Écrire la fonction ***binaire_txt (T)*** qui reçoit en paramètre une liste ***T*** contenant les codes des caractères d'un message. La fonction retourne une chaîne de caractères ***B***, contenant la concaténation des conversions binaires des éléments de ***T***. Chaque élément de ***T*** est converti sur **8 bits**.

Exemple : T = [67, 111, 110, 99, 111, 117, 114, 115, 32, 67, 78, 67, 32, 77, 80, 45, 50, 48, 50, 51]

binaire_txt (T) retourne la chaîne de caractères **B** suivante **: '01000011**01101111**01101110**01100011**0110 1111**01110101**01110010**01110011**00100000**01000011**01001110**01000011**00100000**01001101**01010000**00 101101**00110010**00110000**00110010**00110011**'**

Q.12- Écrire la fonction ***remplissage (B)*** qui reçoit en paramètre la chaîne de caractères ***B***, qui contient la conversion binaire du message. La fonction effectue les opérations suivantes :

- **n** ← longueur (***B***) ;
- ajouter le bit **'1'** à la fin de la chaîne ***B* ;**
- ajouter le bit **'0'** à la fin de ***B*** jusqu'à ce que : la longueur de ***B*** modulo **512** soit égale à **448** ;
- Convertir le nombre **n** en binaire sur **64 bits**, dans une chaîne de caractères, et ajouter cette chaîne à la fin de la chaîne ***B*** ;
- Retourner la nouvelle chaîne **B** *(La taille de **B** est un multiple de **512**)*.

Exemple :

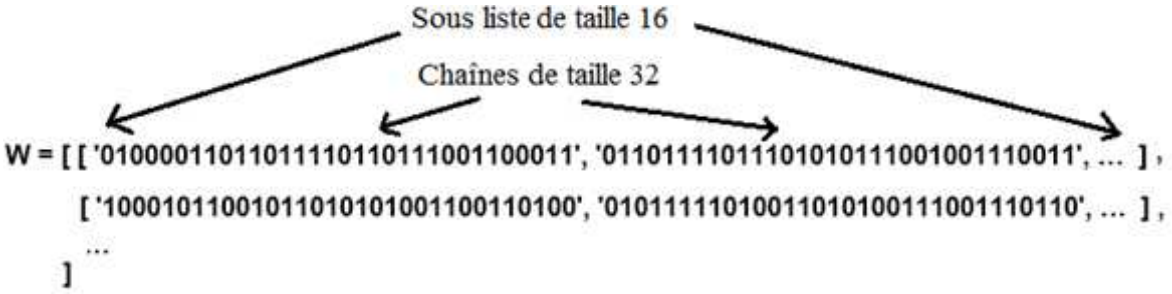
Chaîne ***B*** de taille ***n=160***	Ajouter **'1'**	Ajouter des **'0'**	Ajouter la représentation binaire de ***n*** sur **64** bits
00110100111000101000 ... 00110011	**1**	**00000000… 0000**	**00000 … 000000010100000**

*La taille finale de **B** est un multiple de **512***

Q.13- Écrire la fonction ***decoupe (B)*** qui reçoit en paramètre la chaîne de caractères ***B*** dont la taille est un multiple de **512**. La fonction effectue le découpage suivant, de la chaîne ***B*** :

- La chaîne ***B*** est découpée en ***p*** sous chaînes de taille **512** chacune *(avec **p=taille(B)/512**)* ;
- Chaque sous chaîne de taille **512** est découpée en **16** sous chaînes de taille **32** chacune. Ces **16** sous chaînes sont rangées dans une sous liste ***w*** ;
- ***W*** est une liste qui contiendra les ***p*** sous listes ***w*** de taille **32** chacune ;
- La fonction retourne la liste de listes ***W***.

Exemple :



D- Compression

Pour chaque sous liste **w** de la liste ***W***, on modifie les **8** variables de hachage, à l'aide de l'algorithme suivant :

Ajouter à la fin de la liste **w**, **48** chaînes de taille **32**, initialisées à **'0000000… 000'** *(**0** répété **32** fois)*
Pour i=16 à 63 faire
    S0 ← (w[i-15] rotation_Droite 7) xor (w[i-15] rotation_Droite 18) xor (w[i-15] decalage_Droite 3)
    S1 ← (w[i-2] rotation_Droite 17) xor (w[i-2] rotation_Droite 19) xor (w[i-2] decalage_Droite 10)
    w[i] ← w[i-16] + S0 + w[i-7] + S1
Fin Pour
Initialiser les 8 variables a, b, c, d, e, f, g, h respectivement par h0, h1, h2, h3, h4, h5, h6, h7
Pour i=0 à 63 faire
    S0 ← (e rotation_Droite 6) xor (e rotation_Droite 11) xor (e rotation_Droite 25)
    ch ← (e and f) xor (non e and g)
    tmp1← h + S0 + ch + K[i] + w[i]
    S1 ← (a rotation_Droite 2) xor (a rotation_Droite 13) xor (a rotation_Droite 22)
    maj ← (a and b) xor (a and c) xor (b and c)
    tmp2 ← S1 + maj
    h ← g
    g ← f
    f ← e
    e ← d + tmp1
    d ← c
    c ←b
    b ← a
    a ← tmp1+ tmp2
Fin Pour
Additionner a, b, c, d, e, f, g, h respectivement à h0, h1, h2, h3, h4, h5, h6, h7

Q.14- Écrire la fonction ***compression (W)*** qui effectue le traitement de l'algorithme ci-dessus, pour chaque sous liste ***w*** de la liste ***W***.

E- Calcul du haché final

Le haché final est obtenu par la concaténation des **8** variables globale de hachage. Ainsi, on obtient une chaîne ***H*** de longueur **256** bits. Le haché final est la conversion de ***H*** en hexadécimale : chaque groupement de **4** bits dans ***H*** est remplacé par sa valeur en hexadécimale.

Exemple :

h0 = '00001111001101010000001110110101'
h1 = '10101100010110011101011100010110'
h2 = '00001110110101001100010101111001'
h3 = '11010111010101001000000100000000'
h4 = '11000011001001101100010001111010'
h5 = '10000110011100111111001101100000'
h6 = '00110001111000100111110101000010'
h7 = '11100000100110011110001110100011'

La chaîne de caractères ***H*** est la concaténation des **8** variables globales de hachage **h0**, **h1**, **h2**, **h3**, **h4**, **h5**, **h6** et **h7** :

'0000111100110101000000111011010110101100010110011101011100010110**00001110110101001100010101111001**11010111010101001000000100000000**11000011001001101100010001111010**10000110011100111111001101100000**00110001111000100111110101000010**11100000100110011110001110100011**'**

Ensuite, chaque groupement de **4** bits dans ***H*** est remplacé par sa valeur en hexadécimale :

0000	*1111*	*0011*	*0101*	*0000*	*0011*	*1011*	*0101*	*1010*	*1100*	*…*	*0011*	*1010*	*0011*
0	**f**	**3**	**5**	**0**	**3**	**b**	**5**	**a**	**c**	**…**	**3**	**a**	**3**

En fin, le haché final est composé des **64** caractères hexadécimaux :

'0f3503b5ac59d7160ed4c579d7548100c326c47a8673f36031e27d42e099e3a3'

Q.15- Écrire la fonction ***hache_final ( )*** qui retourne le haché final.

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