Concours National Commun (CNC), Maroc

Session 2023, filière PSI

Informatique

Ce que traite le sujet

- Exercice : les nombres parfaits
- Partie I : base de données et langage SQL
- Partie II : calcul numérique, interpolation de Lagrange
- Partie III : problème, fonctions de hachage

Sujet reproduit sans retouche à partir de la page suivante.

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

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

Épreuve d'Informatique – Session 2023 – Filière PSI

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 parfaits

***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**

*Un entier strictement positifs **x** est **parfait**, si **x** est égal à la somme de ses diviseurs stricts.*

Q.4- Écrire la fonction ***parfait (x)*** qui reçoit en paramètre un entier strictement positifs ***x*** et qui retourne **True** si ***x*** est parfait, sinon, la fonction retourne **False**.

Exemple : ***parfait (28)*** retourne **True** *(28 = 1+2+4+7+14)*

Q.5- Écrire la fonction ***compte_parfaits (L)*** qui reçoit en paramètre une liste ***L*** d'entiers strictement positifs tous différents. La fonction retourne le compte des nombres parfaits dans ***L***.

Exemple : ***compte_parfaits ([25, 28, 123, 10, 6])*** retourne **2** *(28 et 6 sont parfaits)*

Q.6- Écrire la fonction ***liste_parfaits (n)*** qui reçoit en paramètre un entiers ***n*** strictement positif. La fonction retourne une liste ***P*** qui contient tous les nombres parfaits inférieurs strictement à ***n***.

Exemple : ***liste_parfaits (500)*** retourne **[6, 28, 496]**

Partie I : Base de données et langage SQL

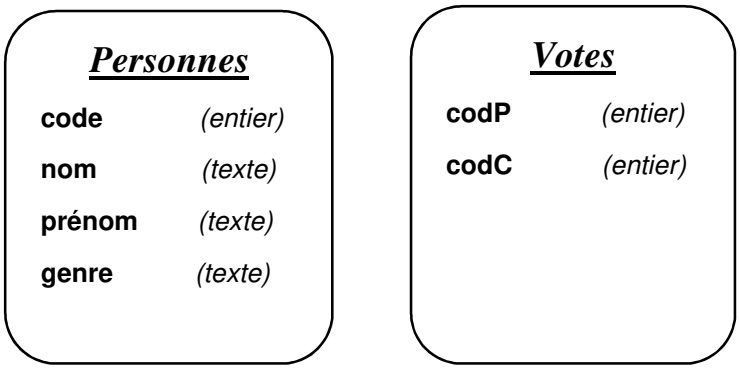
Élections

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 est composé d'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 de Lagrange

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, les **polynômes de Lagrange**, du nom de Joseph-Louis Lagrange, permettent d'interpoler une série de points par un polynôme qui passe exactement par ces points

Étant donnés ***n*** points dans le plan ***(x0, y0)***, ***(x1, y1)***, …, ***(xi, yi)***, …, ***(xn-1, yn-1)***, avec les ***xi*** distincts deux à deux, l'interpolation de Lagrange permet de construire un polynôme ***L*** de degré au plus ***n-1***, et qui passe par ces points : ***L(xi) = yi***

La formule du polynôme ***L*** est la suivante :

$$L(x) = \sum_{j=0}^{n-1} (y_j * (\prod_{i=0 \; et \; i \neq j}^{n-1} \frac{x - x_i}{x_j - x_i}))$$

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 ***xi*** dans ***X*** et ***yi*** dans ***Y*** sont l'abscisse et l'ordonnée du point ***pi*** 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 ***produit (x, X, j)*** qui reçoit en paramètres un réel x, le vecteur ***X*** contenant les abscisses des point à interpoler et un entier ***j*** qui représente un indice dans ***X***. La fonction calcule et retourne la valeur du produit suivant :

$$\prod_{i=0 \; et \; i \neq j}^{n-1} (\frac{x - x_i}{x_j - x_i})$$

Q.3- Écrire la fonction ***L (x, P)*** qui reçoit en paramètres un réel x et la liste ***P*** de tuples qui représentent les points à interpoler. La fonction retourne la valeur de l'expression suivante :

$$\sum_{j=0}^{n-1}(y_j * (\prod_{i=0 \; et \; i \neq j}^{n-1} \frac{x - x_j}{x_i - x_j}))$$

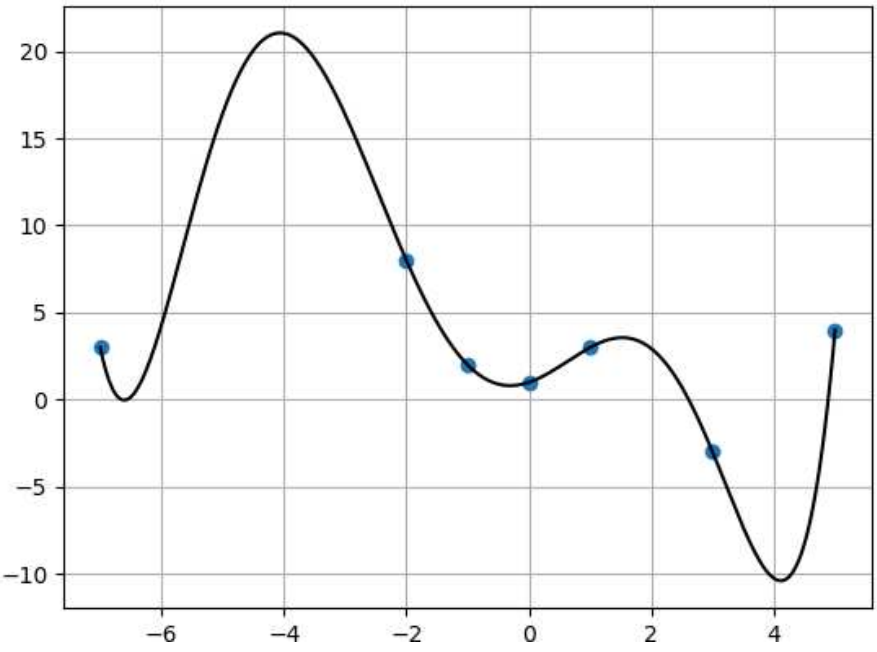
Q.4- Écrire la fonction ***courbe (P)*** qui reçoit en paramètre la liste ***P*** de tuples qui représentent les points à interpoler. La fonction trace la représentation graphique du polynôme interpolateur qui passe par tous les points de ***P***.

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 'MD5'

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 **MD5**.

La fonction de **hachage MD5** a été inventée en 1991. À partir d'un texte de taille maximale $\mathbf{2^{64}}$ bits (c.à.d. **2 Péta-octets**), cette fonction produit un haché de **128** bits, représenté par **32** caractères hexadécimaux.

Exemples :

- Le haché du message **' Concours CNC 2023'** est : **'e7d97a155f941aa3cde034e8a6c365b0'**
- Le haché du message **'Concours CNC PSI-2023'** est : **'aeaaca47dd071eeda95b050ade96f2b7'**

L'objectif de cette partie est d'implémenter l'**algorithme du hachage MD5**, 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 : **'00000010000000000000001000100000'**

Q.3- Écrire la fonction ***OU (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' ou y_i = '1' alors z_i = '1'**
- **Si x_i = '0' et y_i = '0' alors z_i = '0'**

Exemple : OU('01000011010011100100001100100000', '00110010001100000011001000110011') retourne la chaîne : **'01110011011111110011100110100110011'**

Q.4- 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.5- Écrire la fonction ***rotation_gauche (X, p)*** qui reçoit en paramètre une chaîne de caractères ***X*** de taille **32**, contenant une représentation binaire, et un entier ***p***, tel que ***1 ≤ p < 32***. La fonction effectue une rotation des bits de ***X*** de ***p*** positions vers la gauche : *Déplacer les premiers **p** bits de **X** à la fin de **X***.

Exemple : rotation_gauche('01000011010011100100001100100111', 12) retourne la chaîne de caractères : '11100100001100100111**010000110100**'

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 de hachage **MD5** utilise une variable globale ***R***, initialisée par la liste composée des **64** constantes suivantes :

R = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]

L'algorithme hachage **MD5** utilise aussi **4** variables globales de hachage : ***h0***, ***h1***, ***h2*** et ***h3***. Ces variables sont initialisées respectivement par les chaînes de caractères contenant les représentations binaires sur **32** bits, des nombres : **1732584193**, **4023233417**, **2562383102** et **271733878**.

Exemple :

La représentation binaire du nombre **1732584193** sur **32** bits est : **01100111010001010010001100000001**

La variable ***h0*** est initialisée par la chaîne : **'01100111010001010010001100000001'**

Q.7- Écrire la fonction ***binaire (x, t)*** qui reçoit en paramètres deux entiers 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***.

Exemples :

- **binaire (1732584193, 32)** retourne la chaîne de caractères de taille **32 : '01100111010001010010001100000001'**
- **binaire (67, 8)** retourne la chaîne de caractères de taille **8 : '01000011'**

Q.8- Écrire les instructions permettant d'initialiser les ***4*** variables globales de hachage *: **h0**, **h1**, **h2** et **h3***

L'algorithme de hachage **MD5** utilise aussi une liste variable globale ***K***, initialisée par une liste composée de **64** constantes. La liste ***K*** est initialisée par les chaînes de caractères contenant les représentations binaires sur **32** bits, des parties entières des nombres suivants :

$$|sin(i)| * 2^{32} \quad pour\ i = 1 \text{ à } 64$$

Q.9- On suppose que la fonction prédéfinie **sin** est importée :

```
from math import sin
```

Écrire les instructions permettant d'initialiser la variable globale de hachage ***K***.

C- Préparation du message

Le message initial 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 PSI-2023**" sont :

C	**o**	**n**	**c**	**o**	**u**	**r**	**s**		**C**	**N**	**C**		**P**	**S**	**I**	**-**	**2**	**0**	**2**	**3**
67	*111*	*110*	*99*	*111*	*117*	*114*	*115*	*32*	*67*	*78*	*67*	*32*	*80*	*83*	*73*	*45*	*50*	*48*	*50*	*51*

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

T = [67, 111, 110, 99, 111, 117, 114, 115, 32, 67, 78, 67, 32, 80, 83, 73, 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*** contenant 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_txt (T)*** qui reçoit en paramètre une liste ***T*** qui contient 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 en binaire sur **8** bits)*

Exemple :

T = [67, 111, 110, 99, 111, 117, 114, 115, 32, 67, 78, 67, 32, 80, 83, 73, 45, 50, 48, 50, 51]

binaire_txt (T) retourne la chaîne de caractères ***B*** suivante :

'0100001101101111**01101110**01100011**01101111**01110101**01110010**01110011**00100000**01000011**01001110**01000011**00100000**01010000**01010011**01001001**00101101**00110010**00110000**00110010**00110011'**

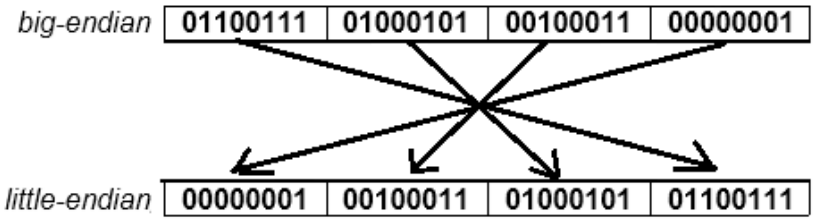
On considère la représentation binaire du nombre **1732584193** sur **32** bits, subdivisée en octets :

01100111	**01000101**	**00100011**	**00000001**

Pour calculer la valeur décimale à partir de cette représentation binaire, on utilise des puissances de **2** dans l'ordre décroissant des exposants, en partant du premier bit (à gauche) jusqu'au dernier bit (à droite) de la représentation binaire.

Cette représentation est dite en **big-endian** : Le premier octet (à gauche) est celui du poids le fort. Le dernier octet (à droite) est celui du poids le plus faible.

La représentation binaire en **little-endian** consiste à inverser l'ordre des octets, de façons à ce que l'octet du poids le plus faible soit en premier (à gauche), et l'octet du poids le plus fort soit à la fin (à droite) :



Q.11- Écrire la fonction ***little_endian (S)*** qui reçoit en paramètre une chaîne de caractères ***S*** dont la taille est un multiple de 8, et qui contient une représentation binaire en big-endian. La fonction effectue la conversion de ***S*** en little-endian.

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 le traitement suivant :

- **n** ← longueur (***B***)
- ajouter le bit "**1**" à la fin de la chaîne ***B***
- ajouter le bit "**0**" à la fin de la chaîne ***B*** jusqu'à ce que la longueur de ***B*** soit égale à **448 (modulo 512)**
- Convertir le nombre ***n*** en binaire sur **64 bits** en little_endian, 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 :

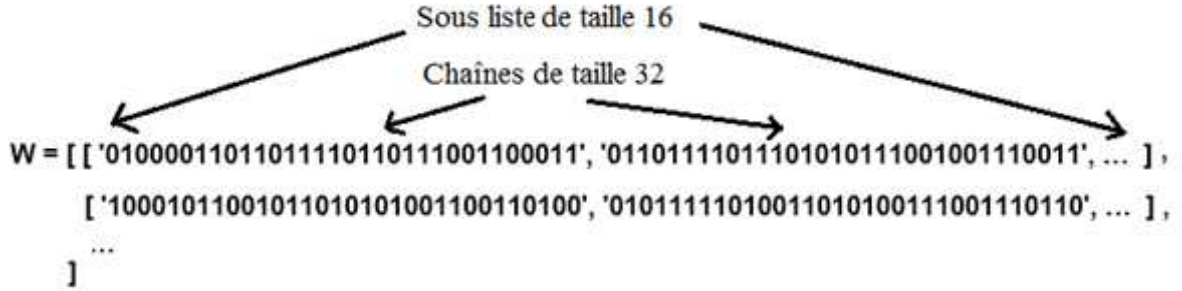
Chaîne ***B*** de taille ***n=168***	Ajouter **'1'**	Ajouter des **'0'**	Ajouter la représentation binaire de ***n*** sur **64** bits
010000110110111101 ... 00110011	**1**	**00000000… 0000**	**00000 … 0000010101000**

*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***, les 4 variables de hachage sont modifiées à l'aide de l'algorithme suivant :

```
Pour chaque w de W faire
        a ← h0
        b ← h1
        c ← h2
        d ← h3
        Pour i=0 à 63 faire
                si 0 ≤ i ≤ 15 alors
                        f ← (b et c) ou ((non b) et d)
                        g ← i
                sinon si 16 ≤ i ≤ 31 alors
                        f ← (d et b) ou ((non d) et c)
                        g ← (5*i + 1) mod 16
                sinon si 32 ≤ i ≤ 47 alors
                        f ← b xor c xor d
                        g ← (3*i + 5) mod 16
                sinon si 48 ≤ i ≤ 63 alors
                        f ← c xor (b ou (non d))
                        g ← (7*i) mod 16
                fin si
                temp ← d
                d ← c
                c ← b
                b ← ((a + f + K[i] + w[g]) rotation_Gauche R[i]) + b
                a ← temp
        Fin pour
        h0 ← h0 + a
        h1 ← h1 + b
        h2 ← h2 + c
        h3 ← h3 + d
Fin pour
```

Q.14- Écrire la fonction ***compression (W)*** qui effectue le traitement de l'algorithme ci-dessus.

E- Calcul du haché final

Le haché final est obtenu par la concaténation des 4 variables globale de hachage ***h0, h1, h2*** *et* ***h3***, chaque variable en little_endian. Ainsi, on obtient une chaîne ***H*** de longueur **128** 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 :

Les 4 variables de hachages ***h0, h1, h2*** *et* ***h3*** en little_endian sont :

h0 = '10101110101010101100101001000111'

h1 = '11011101000001110001111011101101'

h2 = '10101001010110110000010100001010'

h3 = '11011110100101101111001010110111'

H est la concaténation des 4 variables de hachage ***h0***, ***h1***, ***h2*** et ***h3*** :

'10101110101010101100101001000111110111010000011100011110111011010100101011011000001010000101011011110100101101111001010110111'

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

1010	*1110*	*1010*	*1010*	*1100*	*1010*	*0001*	*0111*	*1101*	*1101*	*…*	*0010*	*1011*	*0111*
a	**e**	**a**	**a**	**c**	**a**	**4**	**7**	**d**	**d**	**…**	**2**	**b**	**7**

En fin, le haché final est composé des **32** caractères hexadécimaux :

'aeaaca47dd071eeda95b050ade96f2b7'

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

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

Travailler les chapitres de ce sujet

Prepa Help suit le programme de PSI 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
- 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/psi