Épreuve d'Informatique – Session 2020 – Filière TSI

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.

≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈

Exercice : (4 points)

Somme de nombres entiers

1 pt **Q.1-** Écrire la fonction `somme(n)` qui reçoit en paramètre un entier strictement positif **n**, et qui retourne la valeur de la somme : $\mathbf{1 + 2 + 3 + 4 + \cdots + n}$

Exemple : La fonction **somme(4)** retourne 10

0.5 pt **Q.2-** Déterminer la complexité de la fonction **somme(n)**.

0.75 pt **Q.3-** Écrire la fonction `terme(k)` qui reçoit en paramètre un entier **k** strictement positif, et qui retourne la valeur de l'expression suivante :

$$\frac{\boldsymbol{k}}{\mathbf{1 + 2 + 3 + 4 + \cdots + k}}$$

Exemple : La fonction **terme(4)** retourne le nombre 0.4

0.5 pt **Q.4-** Déterminer la complexité de la fonction **terme(k)**.

1.25 pt **Q.5-** Écrire la fonction `sigma(n)` qui reçoit en paramètre un entier strictement positif **n**, et qui retourne la valeur de la somme suivante :

$$\frac{\mathbf{1}}{\mathbf{1}} + \frac{\mathbf{2}}{\mathbf{1+2}} + \frac{\mathbf{3}}{\mathbf{1+2+3}} + \cdots + \frac{\boldsymbol{n}}{\mathbf{1+2+3+4+\cdots+n}}$$

Partie I : Calcul numérique

Intégration numérique

Méthode de "Simpson"

En analyse numérique, il existe une vaste famille d'algorithmes dont le but principal est d'estimer la valeur numérique de l'intégrale d'une fonction f sur un intervalle $[a, b]$:

$$\int_a^b f(x)dx$$

Ces techniques procèdent en trois phases :

- **a.** Décomposition de l'intervalle $[a, b]$ en sous-intervalles contigus ;
- **b.** Intégration approchée de la fonction sur chaque sous-intervalle ;
- **c.** Sommation des résultats numériques ainsi obtenus.

Méthode de Simpson :

La **méthode de Simpson**, du nom de Thomas Simpson, est une technique qui utilise l'approximation d'ordre 2 de f par un polynôme quadratique P prenant les mêmes valeurs que f aux points d'abscisses a, $m = \frac{a+b}{2}$ et b.

Pour déterminer l'expression de cette parabole (polynôme de degré 2), on utilise l'interpolation lagrangienne. Le résultat peut être mis sous la forme suivante :

$$P(x) = f(a)\frac{(x-m)(x-b)}{(a-m)(a-b)} + f(m)\frac{(x-a)(x-b)}{(m-a)(m-b)} + f(b)\frac{(x-a)(x-m)}{(b-a)(b-m)}$$

Un polynôme étant une fonction très facile à intégrer, on approche l'intégrale de la fonction f sur l'intervalle $[a, b]$, par l'intégrale de P sur ce même intervalle. On a ainsi, la formule simple:

$$\int_a^b f(x)dx \approx \int_a^b P(x)dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Pour appliquer cette méthode d'intégration, on doit découper l'intervalle $[a , b]$ en n sous-intervalles de longueur $h = \frac{b-a}{n}$. Ensuite on applique la formule précédente sur chacun des sous-intervalles, et on effectue la somme des résultats obtenus. En simplifiant la sommation des résultats obtenus, on obtient la formule finale suivante :

$$\int_a^b f(x)dx \approx \frac{h}{6}\left(f(a) + 4f\left(a + \frac{h}{2}\right) + f(b)\right) + \frac{h}{3}\sum_{i=1}^{n-1} f(a+ih) + 2f\left(a + ih + \frac{h}{2}\right)$$

Q.1- Écrire la fonction `somme(f,a,h,n)` qui retourne la valeur de la somme suivante.

$$\sum_{i=1}^{n-1} f(a+ih) + 2f\left(a+ih+\frac{h}{2}\right)$$

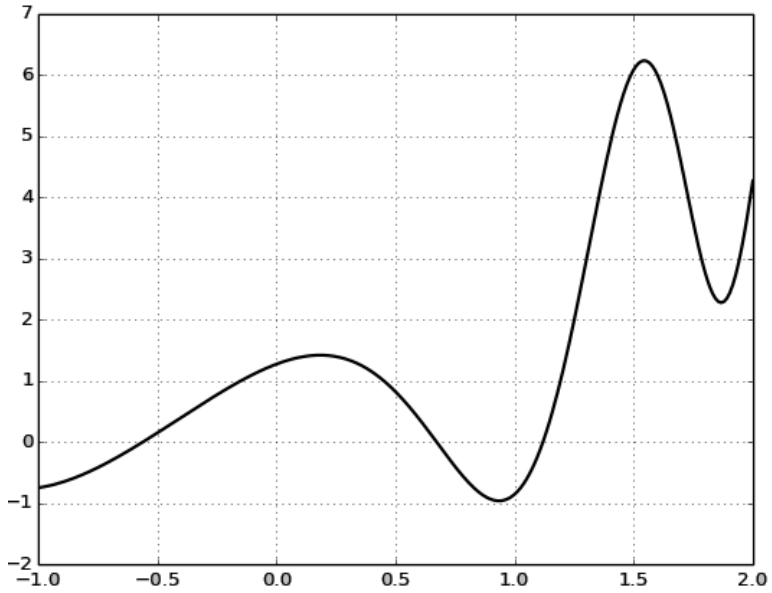
Q.2- Écrire la fonction `integrale_simpson(f,a,b,n)`, qui retourne la valeur de l'intégrale de la fonction f dans l'intervalle $[a, b]$, en utilisant la formule de simpson.

On suppose que les modules **numpy** et **matplotlib.pyplot** sont importés :

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

Q.3.a – Écrire en python, la fonction g définie par : $g(x) = x^2 + 5*\cos(1+x^2)*\sin(e^x) - 1$

Q.3.b – Écrire le programme qui trace la courbe suivante de la fonction g. Le nombre de points générés dans la courbe est : **250**



Q.3.c- Écrire le code python qui utilise la méthode de simpson, et qui calcule et affiche la valeur de l'intégral de la fonction g, dans l'intervalle $[-0.5 , 1.5]$, en utilisant pour nombre de subdivision : $n=10^5$

Partie II : Base de donnés et langage SQL

Classement des candidats

Une grande école d'ingénieurs organise un concours au profit des étudiants (candidats), qui veulent y accéder pour suivre leurs études supérieures. Le concours est composé de plusieurs épreuves : une seule épreuve par matière.

Après le passage des épreuves, une note totale est calculée pour chaque candidat. Ensuite les candidats sont classés dans l'ordre décroissant de la note totale.

*NB : Une note inférieure à **5.0** est une note éliminatoire du classement final. Si un candidat possède au moins une note éliminatoire, alors ce candidat sera exclu du classement final.*

L'école d'ingénieurs utilise une base de données relationnelle composée de trois tables :

Candidats

numéro	*(entier)*
nom	*(texte)*
prénom	*(texte)*

Notes

numC	*(entier)*
codM	*(texte)*
note	*(réel)*

Matières

code	*(texte)*
nom	*(texte)*
coeff	*(entier)*

La table **'Candidats'** contient les numéros, les noms et les prénoms des candidats. Le champ **numéro** est la clé primaire dans cette table.

Exemples :

numéro	nom	prénom
416	Jarfaoui	Hicham
70	Hilal	Samira
162	Senhaji	Amal
23	Bakouri	Ahmed
…	**…**	**…**

La table **'Matières'** contient les codes, les noms et les coefficients des matières. Le champ **code** est la clé primaire dans cette table.

Exemples :

code	nom	coeff
M	Mathématiques	14
P	Physique	10
SI	Sciences de l'ingénieur	6
Ch	Chimie	3
…	**…**	**…**

La table **'Notes'** contient, pour chaque candidat, la note correspondante à chaque matière. Les champs **numC** et **codM** sont deux clés étrangères, qui font respectivement référence, aux champs **numéro** et **code** des tables 'Candidats' et 'Matières'.

Exemples :

numC	codM	note
70	**SI**	14,50
162	**SI**	17,00
416	**SI**	12,25
70	**M**	16,00
162	**M**	13,50
416	**M**	08,35
70	**P**	04,75
162	**P**	11,05
416	**P**	10,10
…	**…**	**…**

Q.1 – Déterminer la clé primaire de la table 'Notes', et justifier votre réponse.

Q.2 – Écrire, en algèbre relationnelle, une requête qui donne les numéros, les noms et prénoms de tous les candidats, qui ne possèdent pas de note éliminatoire, en matière de code **'M'**.

Q.3 – Écrire la requête précédente (**Q.2**) en langage SQL.

Q.4 – Écrire, en langage SQL, une requête qui donne les noms des matières, la note maximale et la note minimale de chaque matière, triés dans l'ordre décroissant de la moyenne des notes de chaque matière.

Q.5 – Écrire, en langage SQL, une requête qui donne le compte des candidats qui sont exclus du classement final.

Q.6– Pour les candidats non exclus du classement final, la **note totale** de chaque candidat est calculée par la formule suivante :

$$\boldsymbol{note\ totale} = \sum(\boldsymbol{note} * \boldsymbol{coefficient})$$

La note d'une matière est multipliée par le coefficient correspondant à cette matière.

Écrire, en langage SQL, une requête qui donne le numéro, le nom, le prénom et la note totale de chaque candidat non exclu, ayant la note totale supérieure strictement à ***1000.0***, triés dans l'ordre décroissant de la note totale.

Partie III : Problème

Transformée de 'Burrows-Wheeler'

La **transformée de Burrows-Wheeler**, couramment désignée par l'acronyme **BWT** (pour *Burrows-Wheeler Transform*) est une technique inventée par Michael Burrows et David Wheeler, et elle a été publiée en 1994. Cette technique permet de réorganiser les données : c'est une transformation qui, à partir d'un texte donné, produit un autre texte contenant exactement les mêmes caractères, mais dans un autre ordre, pour lequel les répétitions de caractères ont tendance à être contiguës.

Le but de cette partie est de réaliser l'algorithme de la transformée de Burrows-Wheeler.

1- Rotation d'une chaîne de caractères

*On considère une chaîne de caractères **T** non vide, et un entier **p** tel que : **0 ≤ p < taille de T**.*

*La **rotation de T d'indice p** est la chaîne de caractères obtenue par la concaténation des deux sous-chaînes suivantes :*

- *La sous-chaîne de **T**, composée de la suite des caractères en partant de la position **p**, jusqu'à la dernière position dans **T** ;*
- *et de la sous-chaîne de **T**, composée de la suite des caractères en partant de la première position de **T**, jusqu'à la position **p-1** dans **T**.*

Exemples :

T = 'daacdacdab'

- La rotation de **T** d'indice **0** est la chaîne : **'daacdacdab'** + '' = **'daacdacdab'**
- La rotation de **T** d'indice **1** est la chaîne : '**aacdacdab**' + 'd' = **'aacdacdab**d**'**
- La rotation de **T** d'indice **2** est la chaîne : '**acdacdab**' + 'da' = **'acdacdab**da**'**
- La rotation de **T** d'indice **9** est la chaîne : '**b**' + 'daacdacda' = **'b**daacdacda**'**

Q.1- Écrire la fonction `rotation(T,p)`, qui reçoit en paramètres une chaîne de caractères **T** non vide, et un entier **p** tel **0 ≤ p < taille de T**. La fonction retourne la rotation de **T** d'indice **p**.

2- Liste des rotations d'une chaîne de caractères

Q.2- Écrire la fonction `liste_rotations(T)`, qui reçoit en paramètre une chaîne de caractères **T** non vide, et qui retourne une liste **R** qui contient toutes les rotations possibles de **T**.

Exemple :

T = 'daacdacdab' (la taille de la chaîne **T** est **10**)

La fonction **liste_rotations (T)** retourne la liste des **10** rotations possibles :

R = ['daacdacdab', 'aacdacdabd', 'acdacdabda', 'cdacdabdaa', 'dacdabdaac', 'acdabdaacd', 'cdabdaacda', 'dabdaacdac', 'abdaacdacd', 'bdaacdacda']

3- Tri de la liste des rotations

Q.3- Écrire la fonction `tri_rotations(R)`, qui reçoit en paramètre la liste **R** qui contient toutes les rotations d'une chaine de caractères. La fonction trie les éléments de **R** dans l'ordre alphabétique.

Exemple :

R= ['daacdacdab', 'aacdacdabd', 'acdacdabda', 'cdacdabdaa', 'dacdabdaac', 'acdabdaacd', 'cdabdaacda', 'dabdaacdac', 'abdaacdacd', 'bdaacdacda']

Après l'appel de la fonction **tri_rotrations (R)**, on obtient la liste triée :

['aacdacdabd', 'abdaacdacd', 'acdabdaacd', 'acdacdabda', 'bdaacdacda', 'cdabdaacda', 'cdacdabdaa', 'daacdacdab', 'dabdaacdac', 'dacdabdaac']

4- Position de la chaîne initiale dans la liste des rotations triée

La liste **R** contient toutes les rotations de la chaine de caractères **T**. On suppose que la liste **R** est triée dans l'ordre alphabétique. La chaîne **T** est un élément de la liste **R** (**T** est la rotation d'indice **0**). Et on veut chercher la position de **T** dans la liste **R** triée. Pour cela, on propose d'utiliser le principe de la **recherche dichotomique**.

La **recherche dichotomique** est un algorithme de complexité logarithmique, qui permet de rechercher la position d'un élément **x** dans une liste **L** triée. Le principe de cet algorithme est le suivant :

1. *Calculer la position **m** du milieu de la liste **L** ;*
2. *Si **L[m] = x**, alors retourner **m** (**m** est la position de **x** dans **L**) ;*
3. *Si **x<L[m]**, alors reprendre l'étape **(1)** en considérant la* première *moitié de la liste **L** ;*
4. *Si **x>L[m]**, alors reprendre l'étape **(1)** en considérant la* deuxième *moitié de la liste **L**.*

Q.4- Écrire la fonction `position(T,R)`, qui reçoit en paramètres une chaîne de caractères **T**, non vide, et la liste **R** de toutes les rotations de **T**, triée dans l'ordre alphabétique. La fonction retourne la position de la chaîne **T** dans la liste triée **R**, en utilisant le principe de la recherche dichotomique.

Exemple :

T = 'daacdacdab'

La liste des rotations de T, triée dans l'ordre alphabétique est :

R = ['aacdacdabd', 'abdaacdacd', 'acdabdaacd', 'acdacdabda', 'bdaacdacda', 'cdabdaacda', 'cdacdabdaa', **'daacdacdab'**, 'dabdaacdac', 'dacdabdaac']

La fonction **position (T, R)** retourne le nombre **7**, qui représente l'indice de la chaîne **T** dans la liste des rotations triée **R**.

5- Transformée de Burrows-Wheeler

La transformée de Burrows-Weeler se contente de réorganiser les caractères d'une chaîne de caractères **T**, de manière à obtenir un autre texte contenant exactement les mêmes caractères de **T**, mais dans un autre ordre, pour lequel les répétitions de caractères ont tendance à être contiguës.

Le principe de la transformée de Burrows-Weeler est le suivant :

1. On construit la liste **R** de toutes les rotations possibles de la chaîne **T** ;
2. On trie les éléments de la liste **R** dans l'ordre alphabétique ;
3. On recherche la position **p** de la chaîne **T** dans la liste **R** triée ;
4. On construit la chaîne **B** composée des derniers caractères des chaines de la liste **R** triée ;
5. Le résultat de la transformée de Burrows-Wheeler est le tuple **(B, p)**.

Exemple :

T = 'daacdacdab'

La liste des rotations de **T**, triée dans l'ordre alphabétique est :

R = ['aacdacdab**d**', 'abdaacdac**d**', 'acdabdaac**d**', 'acdacdabd**a**', 'bdaacdacd**a**', 'cdabdaacd**a**', 'cdacdabda**a**', 'daacdacda**b**', 'dabdaacda**c**', 'dacdabdaa**c**']

La position de **T** dans **R** est : **p=7**.

La chaîne composée des derniers caractères des éléments de la liste **R** triée est : **B = 'dddaaaabcc'**.

Le résultat de la transformée de Burrows-Wheeler est le tuple : **('dddaaaabcc'** , **7)**

Q.5- Écrire la fonction **`BWT(T)`**, qui reçoit en paramètre une chaîne de caractères **T** non vide, et qui retourne le tuple **(B, p)**.

Exemple :

T = 'daacdacdab'

La fonction **BWT (T)** retourne le tuple : **('dddaaaabcc'** , **7)**

≈ ≈ ≈ ≈ ≈ ≈ FIN DE L'ÉPREUVE ≈ ≈ ≈ ≈ ≈ ≈