Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Réseaux & Système

Examen – Janvier 2020

Durée : 2h — Documents autorisés

Programmation Python — (8 points)

1 – 8pts La « *distance de Hamming* » est une **métrique** permettant de comparer deux séquences de bits de même longueur : le **nombre de positions** pour laquelle la valeur des deux bits correspondant est **différente**.

La distance de Hamming de deux séquences de bits a et b est notée $d(a,b)$.

Pour déterminer $d(a,b)$, on calcule $a \oplus b$ et on compte le nombre de bit à 1 dans le résultat.

Exemple : soient deux séquences de bits `1101 1001` et `1001 1101` :
▷ $11011001 \oplus 10011101 = 01000100$, où $\oplus$ est l'opération « *xor* » ;
▷ la séquence résultat `0100 0100` contenant deux 1, la distance de Hamming est de 2.
$\Longrightarrow d(11011001, 10011101) = 2$

On appelle la « *distance minimale de Hamming* », la distance de Hamming **la plus petite** obtenue entre **toutes les paires de séquence** appartenant à un **ensemble de séquences de même taille**.

a. Soient l'ensemble de séquences de bits : $1110010, 1001001, 1001111, 0101100$ *(1pt)*
Calculez la distance minimale de Hamming pour cet ensemble.

b. Sous Python, l'opération « *xor* » est notée `^` et s'utilise de la manière suivante : *(1pt)*

```
xterm
>>> 12^3
15
```

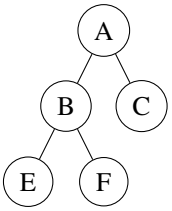
Expliquez comment, en Python, vous allez calculer le « *xor* » entre deux séquences de bits.

c. Écrivez une fonction Python acceptant deux séquences de bits en argument et retournant la distance de Hamming entre ces deux séquences de bits. *(2pts)*

d. Écrivez un programme Python utilisant la fonction précédente pour déterminer la « *distance minimale de Hamming* » d'un ensemble de séquence de bits donné sous forme d'une liste. *(2pts)*

e. Écrivez un programme Python qui pour une séquence de bits donnée quelconque, retourne la séquence la plus proche d'un ensemble E de séquences connues (la séquence n'appartient pas forcément à E). *(2pts)*

2 – 3pts On construit une représentation d'un **arbre binaire** sous forme de listes imbriquées :
`nœud = [ tête, enfant1, enfant2]` (un nœud vide est indiqué par une liste vide).




```
1  arbre = ['A',
2           ['B',
3              ['E', [], []],
4              ['F', [], []]
5           ], # fin de B
6           ['C',
7              [],
8              []
9           ] # fin de C
10       ]
```

Écrire un programme Python réalisant l'affichage de chaque nœud de l'arbre.

Unix — (2 points)

3 – 2pts a. Pour un processeur 64 bits sous Linux, la taille d'une page est de 4096 octets : de combien de pages dispose-t-on ? *(1pt)*

b. Est-ce que le décalage d'un octet de l'adresse de départ d'un programme en mémoire centrale est grave lors de son exécution par le processeur ? Expliquez ce qui arrive. *(1pt)*

▬▬▬ Réseaux — (5 points)

4 – 5pts

a. On veut concevoir un **serveur TCP** qui reçoit une valeur en notation **hexadécimale** et renvoie la valeur **entière** correspondante. *(1pt)*
Que faut-il « *partager* » avec le client qui l'utilisera ?

b. Écrire un programme Python TCP serveur qui **pour chaque connexion reçue**, réalise la conversion valeur hexadécimale vers valeur entière (une seule conversion par connexion). *(3pts)*

c. Si on veut étendre les fonctionnalités du serveur pour réaliser l'**opération inverse** « *valeur entière* $\Longrightarrow$ *valeur hexadécimale* », comment faut-il le faire pour que cela marche ? *(1pt)*