Métropole - mars 2021 - sujet 1

Exercice 1 (Arbre binaire de recherche et POO - 4 points) Dans cet exercice, les arbres binaires de recherche ne peuvent pas comporter plusieurs fois la même clé. De plus, un arbre binaire de recherche limité à un nœud a une hauteur de 0.
On considère l'arbre binaire de recherche représenté ci-dessous (figure 1), où `val` représente un entier :

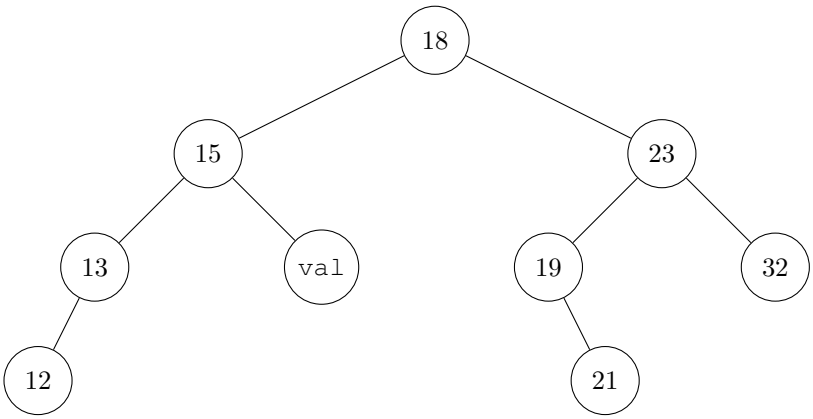


FIGURE 1

1. (a) Donner le nombre de feuilles de cet arbre et préciser leur valeur (étiquette).
   (b) Donner le sous arbre-gauche du nœud 23.
   (c) Donner la hauteur et la taille de l'arbre.
   (d) Donner les valeurs entières possibles de `val` pour cet arbre binaire de recherche.

On suppose dans la suite que `val` *est égal à* 16.

2. On rappelle qu'un parcours infixe depuis un nœud consiste, dans l'ordre, à faire un parcours infixe sur le sous arbre-gauche, afficher le nœud puis faire un parcours infixe sur le sous-arbre droit. Dans le cas d'un parcours suffixe, on fait un parcours suffixe sur le sous-arbre gauche puis un parcours suffixe sur le sous-arbre droit, avant d'afficher le nœud.
   (a) Donner les valeurs d'affichage des nœuds dans le cas du parcours infixe de l'arbre.
   (b) Donner les valeurs d'affichage des nœuds dans le cas du parcours suffixe de l'arbre.
3. On considère la classe `Noeud` définie de la façon suivante en `Python` :

```
class Noeud():
    def __init__(self, v):
        self.ag = None
        self.ad = None
        self.v = v

    def insere(self, v):
        n = self
        est_insere = False
        while not est_insere :
            if v == n.v:
                est_insere = True                    } bloc 1
            elif v < n.v:
                if n.ag != None:                     }
                    n = n.ag                         }
                else:                                } bloc 2
                    n.ag = Noeud(v)                  }
                    est_insere = True                }
            else:
                if n.ad != None:                     }
                    n = n.ad                         }
                else:                                } bloc 3
                    n.ad = Noeud(v)                  }
                    est_insere = True                }

    def insere_tout(self, vals):
        for v in vals:
            self.insere(v)
```

(a) Représenter l'arbre construit suite à l'exécution de l'instruction suivante :

```
racine = Noeud(18)
racine.insere_tout([12, 13, 15, 16, 19, 21, 32, 23])
```

(b) Ecrire les deux instructions permettant de construire l'arbre de la figure 1. On rappelle que le nombre `val` est égal à 16.

(c) On considère l'arbre tel qu'il est présenté sur la figure 1. Déterminer l'ordre d'exécution des blocs (repérés de 1 à 3) suite à l'application de la méthode `insere(19)` au nœud racine de cet arbre.

4. Ecrire une méthode `recherche` qui prend en argument un entier `v` et renvoie la valeur `True` si cet entier est une étiquette de l'arbre et `False` sinon.

Exercice 2 (Processus et opérateurs booléens - 4 points)

Partie A

Cette partie est un questionnaire à choix multiples (QCM). Pour chacune des questions, une seule des quatre réponses est exacte. Le candidat indiquera sur sa copie le numéro de la question et la lettre correspondant à la réponse exacte. Aucune justification n'est demandée. Une réponse fausse ou une absence de réponse n'enlève aucun point.

1. Parmi les commandes ci-dessous, laquelle permet d'afficher les processus en cours d'exécution ?
   (a) `dir`
   (b) `ps`
   (c) `man`
   (d) `ls`
2. Quelle abréviation désigne l'identifiant d'un processus dans un système d'exploitation de type UNIX ?
   (a) PIX
   (b) SIG
   (c) PID
   (d) SID
3. Comment s'appelle la gestion du partage du processeur entre différents processus ?
   (a) L'interblocage
   (b) L'ordonnancement
   (c) La planification
   (d) La priorisation
4. Quelle commande permet d'interrompre un processus dans un système d'exploitation de type UNIX ?
   (a) `stop`
   (b) `interrupt`
   (c) `end`
   (d) `kill`

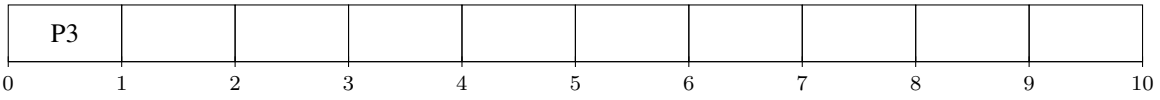
Partie B

1. Un processeur choisit à chaque cycle d'exécution le processus qui doit être exécuté. Le tableau ci-dessous donne pour trois processus P1, P2 et P3 :
   - la durée d'exécution (en nombre de cycles),
   - l'instant d'arrivée sur le processeur (exprimé en nombre de cycles à partir de 0),
   - le numéro de priorité.

   Le numéro de priorité est d'autant plus petit que la priorité est grande. On suppose qu'à chaque instant, c'est le processus qui a le plus petit numéro de priorité qui est exécuté, ce qui peut provoquer la suspension d'un autre processus, lequel reprendra lorsqu'il sera le plus prioritaire.

Processus	Durée d'exécution	Instant d'arrivée	Numéro de priorité
P1	3	3	1
P2	3	2	2
P3	4	0	3

   Reproduire le tableau ci-dessous sur la copie et indiquer dans chacune des cases le processus exécuté à chaque cycle.



2. On suppose maintenant que les trois processus précédents s'exécutent et utilisent une ou plusieurs ressources parmi R1, R2 et R3. Parmi les scénarios suivants, lequel provoque un interblocage ? Justifier.

Scénario 1
P1 acquiert R1
P2 acquiert R2
P3 attend R1
P2 libère R2
P2 attend R1
P1 libère R1

Scénario 2
P1 acquiert R1
P2 acquiert R3
P3 acquiert R2
P1 attend R2
P2 libère R3
P3 attend R1

Scénario 3
P1 acquiert R1
P2 acquiert R2
P3 attend R2
P1 attend R2
P2 libère R2
P3 acquiert R2

Partie C

Dans cette partie, pour une meilleure lisibilité, des espaces sont placées dans les écritures binaires des nombres. Il ne faut pas les prendre en compte dans les calculs.

Pour chiffrer un message, une méthode, dite du masque jetable, consiste à le combiner avec une chaîne de caractères de longueur comparable. Une implémentation possible utilise l'opérateur `XOR` (ou exclusif) dont voici la table de vérité :

a	**b**	**a XOR b**
0	0	0
0	1	1
1	0	1
1	1	0

Dans la suite, les nombres écrits en binaire seront précédés du préfixe `0b`.

1. Pour chiffrer un message, on convertit chacun de ses caractères en binaire (à l'aide du format Unicode), et on réalise l'opération `XOR` bit à bit avec la clé. Après conversion en binaire, et avant que l'opération `XOR` bit à bit avec la clé n'ait été effectuée, Alice obtient le message suivant :

```
m = 0b 0110 0011 0100 0110
```

   (a) Le message `m` correspond à deux caractères codés chacun sur 8 bits : déterminer quels sont ces caractères. On fournit pour cela la table ci-dessous qui associe à l'écriture hexadécimale d'un octet le caractère correspondant (figure 2). Exemple de lecture : le caractère correspondant à l'octet codé $4A$ en hexadécimal est la lettre `J`.

	0	1	2	3	4	5	6	7	8	9	*A*	*B*	*C*	*D*	*E*	*F*
2	space	!	"	#	$	%	&	'	(	)	*	+	,	−	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	0
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

FIGURE 2

   (b) Pour chiffrer le message d'Alice, on réalise l'opération `XOR` bit à bit avec la clé suivante :

```
k = 0b 1110 1110 1111 0000
```

   Donner l'écriture binaire du message obtenu.

2. (a) Dresser la table de vérité de l'expression booléenne `(a XOR b) XOR b`

   (b) Bob connaît la chaîne de caractères utilisée par Alice pour chiffrer le message. Quelle opération doit-il réaliser pour déchiffrer son message ?

Exercice 3 (SQL - 4 points)

L'énoncé de cet exercice utilise les mots du langage SQL suivants : `SELECT FROM`, `WHERE`, `JOIN ON`, `INSERT INTO VALUES`, `UPDATE`, `SET`, `DELETE`, `COUNT`, `AND` et `OR`.

Pour la gestion des réservations clients, on dispose d'une base de données nommée « gare » dont le schéma relationnel est le suivant :

`Train (numT, provenance, destination, horaireArrivee, horaireDepart)`

`Reservation (numR, nomClient, prenomClient, prix, #numT)`

Les attributs soulignés sont des clés primaires. L'attribut précédé de `#` est une clé étrangère.
La clé étrangère `Reservation.numT` fait référence à la clé primaire `Train.numT`.
Les attributs `horaireDepart` et `horaireArrivee` sont de type `TIME` et s'écrivent selon le format `hh:mm`, où `hh` représente les heures et `mm` les minutes.

1. Quel nom générique donne-t-on aux logiciels qui assurent, entre autres, la persistance des données, l'efficacité de traitement des requêtes et la sécurisation des accès pour les bases de données ?
2. (a) On considère les requêtes SQL suivantes :

```
DELETE FROM Train WHERE numT = 1241
DELETE FROM Reservation WHERE numT = 1241
```

   Sachant que le train n°1241 a été enregistré dans la table `Train` et que des réservations pour ce train ont été enregistrées dans la table `Reservation`, expliquer pourquoi cette suite d'instructions renvoie une erreur.

   (b) Citer un cas pour lequel l'insertion d'un enregistrement dans la table `Reservation` n'est pas possible.
3. Ecrire des requêtes SQL correspondant à chacune des instructions suivantes :
   (a) Donner tous les numéros des trains dont la destination est « Lyon ».
   (b) Ajouter une réservation n°1307 de 33 € pour M. Alan Turing dans le train n°654.
   (c) Suite à un changement, l'horaire d'arrivée du train n°7869 est programmé à 08h11. Mettre à jour la base de données en conséquence.
4. Que permet de déterminer la requête suivante ?

```
def hauteur(self):
    if self.gauche == None and self.droit == None:
        return 1
    if self.gauche == None:
        return 1+self.droit.hauteur()
    elif self.droit == None:
        return 1+self.gauche.hauteur()
    else:
        hg = self.gauche.hauteur()
        hd = self.droit.hauteur()
        if hg > hd:
            return hg+1
        else:
            return hd+1
```

5. Ecrire la requête qui renvoie les destinations et les prix des réservations effectuées par Grace Hopper.

Exercice 4 (Tri fusion et « diviser pour régner » - 4 points)

1. (a) Quel est l'ordre de grandeur du coût, en nombre de comparaisons, de l'algorithme de tri fusion pour une liste de longueur n ?

   (b) Citer le nom d'un autre algorithme de tri. Donner l'ordre de grandeur de son coût, en nombre de comparaisons, pour une liste de longueur n. Comparer ce coût à celui du tri fusion. Aucune justification n'est attendue.

L'algorithme de tri fusion utilise deux fonctions `moitie_gauche` et `moitie_droite` qui prennent en argument une liste `L` et renvoient respectivement :

- la sous-liste de `L` formée des éléments d'indice strictement inférieur à `len(L)//2` ;
- la sous-liste de `L` formée des éléments d'indice supérieur ou égal à `len(L)//2`.

On rappelle que la syntaxe `a//b` désigne la division entière de `a` par `b`. Par exemple,

```
>>> L = [3, 5, 2, 7, 1, 9, 0]
>>> moitie_gauche(L)
[3, 5, 2]
>>> moitie_droite(L)
[7, 1, 9, 0]
```

```
>>> M = [4, 1, 11, 7]
>>> moitie_gauche(M)
[4, 1]
>>> moitie_droite(M)
[11, 7]
```

L'algorithme utilise aussi une fonction `fusion` qui prend en argument deux listes triées `L1` et `L2` et renvoie une liste `L` triée et composée des éléments de `L1` et `L2`

On donne ci-dessous le code `Python` d'une fonction récursive `tri_fusion` qui prend en argument une liste `L` et renvoie une nouvelle liste triée formée des éléments de `L`.

```
def tri_fusion(L):
    n = len(L)
    if n<=1:
        return L
    print(L)
    mg = moitie_gauche(L)
    md = moitie_droite(L)
    L1 = tri_fusion(mg)
    L2 = tri_fusion(md)
    return fusion(L1, L2)
```

2. Donner la liste des affichages produits par l'appel `tri_fusion([7, 4, 2, 1, 8, 5, 6, 3])`.

On s'intéresse désormais à différentes fonctions appelées par `tri_fusion`, à savoir `moitie_droite` et `fusion`.

3. Ecrire la fonction `moitie_droite`.
4. On donne ci-dessous une version incomplète de la fonction `fusion`.

```
1  def fusion(L1, L2):
2      L = []
3      n1 = len(L1)
4      n2 = len(L2)
5      i1 = 0
6      i2 = 0
7      while i1 < n1 or i2 < n2:
8          if i1 >= n1:
9              L.append(L2[i2])
10             i2 = i2 + 1
11         elif i2 >= n2:
12             L.append(L1[i1])
13             i1 = i1 + 1
14         else:
15             e1 = L1[i1]
16             e2 = L2[i2]
17
18
19
20
21
22
23     return L
```

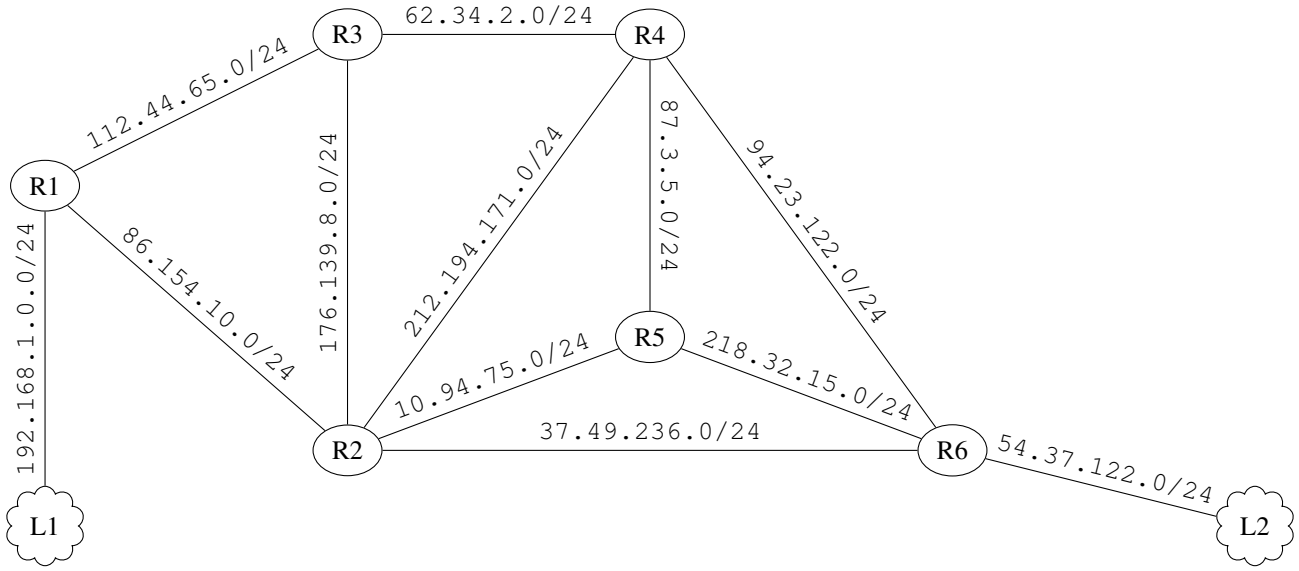
Dans cette fonction, les entiers `i1` et `i2` représentent respectivement les indices des éléments des listes `L1` et `L2` que l'on souhaite comparer :

- Si aucun des deux indices n'est valide, la boucle `while` est interrompue ;
- Si `i1` n'est plus un indice valide, on va ajouter à `L` les éléments de `L2` à partir de l'indice `i2` ;
- Si `i2` n'est plus un indice valide, on va ajouter à `L` les éléments de `L1` à partir de l'indice `i1` ;
- Sinon, le plus petit élément non encore traité est ajouté à `L` et on décale l'indice correspondant.

Ecrire sur la copie les instructions manquantes des lignes `17` à `22` permettant d'insérer dans la liste `L` les éléments des listes `L1` et `L2` par ordre croissant.

Exercice 5 (Routage - 4 points)

On représente ci-dessous un réseau dans lequel R1, R2, R3, R4, R5 et R6 sont des routeurs. Le réseau local L1 est relié au routeur R1 et le réseau local L2 au routeur R6.



Dans cet exercice, les adresses IP sont composées de 4 octets, soit 32 bits. Elles sont notées `X1.X2.X3.X4`, où `X1`, `X2`, `X3` et `X4` sont les valeurs des 4 octets, convertis en notation décimale.

La notation `X1.X2.X3.X4/n` signifie que les `n` premiers bits de poids forts de l'adresse IP représentent la partie « réseau », les bits suivants représentent la partie « hôte ».

Toutes les adresses des hôtes connectés à un réseau local ont la même partie réseau et peuvent donc communiquer directement. L'adresse IP dont tous les bits de la partie « hôte » sont à 0 est appelée « adresse du réseau ».

On donne également des extraits de la table de routage des routeurs R1 à R5 dans le tableau suivant :

Routeur	Réseau destinataire	Passerelle	Interface
R1	`54.37.122.0/24`	`86.154.10.1/24`	`86.154.10.56/24`
R2	`54.37.122.0/24`	`37.49.236.22/24`	`37.49.236.23/24`
R3	`54.37.122.0/24`	`62.34.2.8/24`	`62.34.2.9/24`
R4	`54.37.122.0/24`	`94.23.122.10/24`	`94.23.122.11/24`
R5	`54.37.122.0/24`	`218.32.15.1/24`	`218.32.15.2/24`

1. Un paquet part du réseau local L1 à destination du réseau local L2.
   (a) En utilisant l'extrait de la table de routage de R1, vers quel routeur R1 envoit-il ce paquet : R2 ou R3 ? Justifier.
   (b) A l'aide des extraits de tables de routage ci-dessus, nommer les routeurs traversés par ce paquet, lorsqu'il va du réseau L1 au réseau L2.
2. La liaison entre R1 et R2 est rompue.
   (a) Sachant que ce réseau utilise le protocole RIP (distance en nombre de sauts), donner l'un des deux chemins possibles que pourra suivre un paquet allant de L1 vers L2.
   (b) Dans les extraits de tables de routage ci-dessus, pour le chemin de la question 2.a, quelle(s) ligne(s) sera (seront) modifiée(s) ?
3. On a rétabli la liaison entre R1 et R2. Par ailleurs, pour tenir compte du débit des liaisons, on décide d'utiliser le protocole OSPF (distance liée au coût minimal des liaisons) pour effectuer le routage. Le coût des liaisons entre routeurs est donné ci-dessous :

Liaison	R1-R2	R1-R3	R2-R3	R2-R4	R2-R5	R2-R6	R3-R4	R4-R5	R4-R6	R5-R6
Coût	100	100	?	1	10	10	10	1	10	1

   (a) Le coût C d'une liaison est donné ici par la formule

$$C = \frac{10^8}{\text{BP}},$$

   où BP est la bande passante de la connexion en bps (bit par seconde). Sachant que la bande passante de la liaison R2-R3 est de 10 Mbps, calculer le coût correspondant.
   (b) Déterminer le chemin parcouru par un paquet partant du réseau L1 et arrivant au réseau L2, en utilisant le protocole OSPF.
   (c) Indiquer pour quel(s) routeur(s) l'extrait de la table de routage sera modifié pour un paquet à destination de L2, avec la métrique OSPF.