BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

BAC BLANC 2023

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice et du dictionnaire n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 6 pages numérotées de 1 à 6.

Le sujet est composé de 3 exercices indépendants.

Exercice 1 : (6pt) *L'exercice porte sur les bases de données et les types construits de données.*
On pourra utiliser les mots clés SQL suivants :

`AND`, `SELECT`, `FROM`, `WHERE`, `JOIN`, `INSERT INTO`, `VALUES`, `COUNT`, `ORDER BY`, `OR`, `ON`, `SET`, `UPDATE`.

On étudie une base de données permettant la gestion de l'organisation d'un festival de musique de jazz, dont voici le schéma relationnel comportant trois relations :

- la relation `groupes(idgrp, nom, style, nb_pers)`
- la relation `musiciens(idmus, nom, prenom, instru, #idgrp)`
- la relation `concerts(idconc, scene, heure_debut, heure_fin, #idgrp)`

Dans ce schéma relationnel :

- les clés primaires sont soulignées ;
- les clés étrangères sont précédées d'un #.

Ainsi `concerts.idgrp` est une clé étrangère faisant référence à `groupes.idgrp`.
Voici un extrait des tables `groupes`, `musiciens` et `concerts` :

extrait de `groupes`

idgrp	nom	style	nb_pers
12	'Weather Report'	'Jazz Fusion'	5
25	'The 3 Sounds'	'Soul Jazz'	4
87	'Return to Forever'	'Jazz Fusion'	8
96	'The Jazz Messenger'	'Hard Bop'	3

extrait de `musiciens`

idmus	nom	prenom	instru	idgrp
12	'Garrett'	'Kenny'	'saxophone'	96
13	'Garrett'	'Kenny'	'flute'	25
58	'Corea'	'Chick'	'piano'	87
97	'Clarke'	'Stanley'	'basse'	87

extrait de `concerts`

idconc	scene	heure_debut	heure_fin	idgrp
10	1	'20h00'	'20h45'	12
24	2	'20h00'	'20h45'	25
36	1	'21h00'	'22h00'	96
45	3	'18h00'	'18h30'	87

1) Citer les attributs de la table `groupes`.
2) Justifier que l'attribut `nom` de la table `musiciens` ne peut pas être une clé primaire.

Solution :

1) Les attributs sont `idgrp`, `nom`, `style` et `nb_pers`.
2) Puisque des musiciens peuvent avoir le même nom, ce qui est le cas ici, `nom` ne peut pas être une clé primaine.

3) En s'appuyant uniquement sur l'extrait des tables fourni, écrire ce que renvoie la requête :

```
SELECT nom FROM groupes WHERE style = 'Jazz Fusion';
```

Solution : On va obtenir `'Weather Report'` et `'Return to Forever'`.

4) Le concert dont l'`idconc` est 36 finira à 22h30 au lieu de 22h00. Recopier sur la copie et compléter la requête SQL ci-dessous permettant de mettre à jour la relation concerts pour modifier l'horaire de fin de ce concert.

```
UPDATE concerts SET ... WHERE ...;
```

Solution : `UPDATE concerts SET heure_fin='22h30' WHERE idconc=36;`

5) Donner une requête SQL permettant de récupérer le nom de tous les groupes qui jouent sur la scène 1.

Solution :

```
SELECT nom FROM groupes
JOIN concerts ON groupes.idgrp=concerts.idgrp
WHERE scene=1;
```

6) Fournir une requête SQL permettant d'ajouter dans la relation `groupes` le groupe `'Smooth Jazz Fourplay'`, de style `'Free Jazz'`, composé de 4 membres. Ce groupe aura un `idgrp` de 17.

Solution :

```
INSERT INTO groupes VALUES (17, 'Smooth Jazz Fourplay', 'Free Jazz', 4);
```

Les données sont ensuite récupérées pour être analysées par la société qui produit les festivals de musique. Pour ce faire, elle utilise la programmation en Python afin d'effectuer certaines opérations plus complexes.
Elle stocke les données relatives aux musiciens sous forme d'un tableau de dictionnaires dans laquelle a été ajouté le nombre de concerts effectués par chaque musicien :

```
>>> print(musiciens)
  [{'idmus': 12, 'nom': 'Garrett', 'prenom': 'Kenny',
    'instru': 'saxophone alto', 'idgrp' : 96, 'nb_concerts': 5},
   {'idmus': 13, 'nom': 'Garrett', 'prenom': 'Kenny',
    'instru': 'flute', 'idgrp' : 25, 'nb_concerts': 3},
   {'idmus': 58, 'nom': 'Corea', 'prenom': 'Chick',
    'instru': 'piano', 'idgrp' : 87, 'nb_concerts': 4},
   {'idmus': 97, 'nom': 'Clarke', 'prenom': 'Stanley',
    'instru': 'basse', 'idgrp' : 87, 'nb_concerts': 4},
   ...
  ]
```

7) Écrire la fonction `recherche_nom` ayant pour unique paramètre un tableau de dictionnaires (comme `musiciens` présenté précédemment) renvoyant un tableau contenant les `idmus` de tous les musiciens ayant participé à au moins 4 concerts.

Solution :

```
def recherche_nom(musiciens):
    resultat = []
    for m in musiciens:
        if m['nb_concerts'] >= 4:
            resultat.append(m['idmus'])
    return resultat
```

Exercice 2 : (12pt) *L'exercice porte sur les arbres binaires de recherche et la programmation objet.*
Dans un entrepôt de e-commerce, un robot mobile autonome exécute successivement les tâches qu'il reçoit tout au long de la journée.
La mémorisation et la gestion de ces tâches sont assurées par une structure de données.

1) Dans l'hypothèse où les tâches devraient être extraites de cette structure (pour être exécutées) dans le même ordre qu'elles ont été mémorisées, préciser si ce fonctionnement traduit le comportement d'une file ou d'une pile. Justifier.

Solution : Puisque la première tâche mémorisée doit aussi être la première à être exécutée, c'est une structure FIFO, donc une file.

En réalité, selon l'urgence des tâches à effectuer, on associe à chacune d'elles, lors de la mémorisation, un indice de priorité (nombre entier) distinct : il n'y a pas de valeur en double.
Plus cet indice est faible, plus la tâche doit être traitée prioritairement.
La structure de données retenue est assimilée à un arbre binaire de recherche (ABR) dans lequel chaque nœud correspond à une tâche caractérisée par son indice de priorité.

Rappel : Dans un arbre binaire de recherche, chaque nœud est caractérisé par une valeur (ici l'indice de priorité), telle que chaque nœud du sous-arbre gauche a une valeur strictement inférieure à celle du nœud considéré, et que chaque nœud du sous-arbre droit possède une valeur strictement supérieure à celle-ci.
Cette structure de données présente l'avantage de mettre efficacement en œuvre l'insertion ou la suppression de nœuds, ainsi que la recherche d'une valeur.

Par exemple, le robot a reçu successivement, dans l'ordre, des tâches d'indice de priorité 12, 6, 10, 14, 8 et 13. En partant d'un arbre binaire de recherche vide, l'insertion des différentes priorités dans cet arbre donne la figure ci-contre.

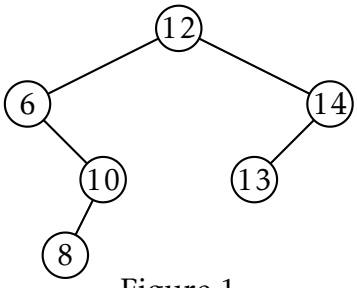


Figure 1

2) En utilisant le vocabulaire couramment utilisé pour les arbres, préciser le terme qui correspond :
   a) au nombre de tâches restant à effectuer, c'est-à-dire le nombre total de nœuds de l'arbre ;
   Solution : C'est la **taille** de l'arbre.
   b) au nœud représentant la tâche restant à effectuer la plus ancienne ;
   Solution : C'est la **racine** qui est la première à avoir été ajoutée.
   c) à un nœud représentant la dernière tâche mémorisée (la plus récente).
   Solution : C'est une **feuille**.
3) Lorsque le robot reçoit une nouvelle tâche, on déclare un nouvel objet, instance de la classe `Noeud`, puis on l'insère dans l'arbre binaire de recherche (instance de la classe `ABR`) du robot. Ces 2 classes sont définies comme suit :

```
1  class Noeud:
2      def __init__(self, tache, indice):
3          self.tache = tache    #ce que doit accomplir le robot
4          self.indice = indice  #indice de priorité (int)
5          self.gauche = ABR()   #sous-arbre gauche vide (ABR)
6          self.droite = ABR()   #sous-arbre droit vide (ABR)
7
8  class ABR:
9      #arbre binaire de recherche initialement vide
10     def __init__(self):
11         self.racine = None #arbre vide
12         #Remarque : si l'arbre n'est pas vide, racine est une instance de la classe Noeud
13
14     def est_vide(self):
15         #renvoie True si l'arbre autoréférencé est vide, False sinon
16         return self.racine == None
17
18     def insere(self, nouveau_noeud):
19         #insere un nouveau noeud, instance de la classe Noeud, dans l'ABR
20         if self.est_vide():
21             self.racine = nouveau_noeud
22         elif self.racine.indice ____ nouveau_noeud.indice:
23             self.racine.gauche.insere(nouveau_noeud)
24         else:
25             self.racine.droite.insere(nouveau_noeud)
```

   a) Donner les noms des attributs de la classe `Noeud`.

Solution : Les attributs sont `tache`, `indice`, `gauche` et `droite`.

b) Expliquer en quoi la méthode `insere` est dite récursive et justifier rapidement qu'elle se termine.

Solution : La méthode `insere` s'appelle elle-même (lignes 23 et 25). Elle est donc récursive.

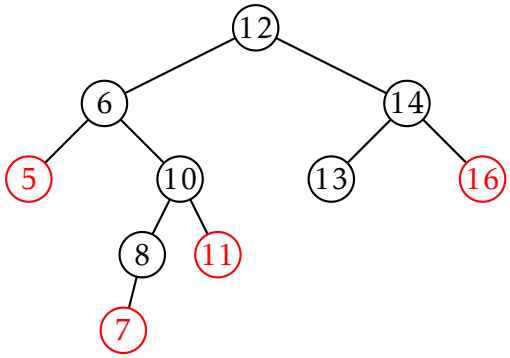
Puisqu'à chaque étape on descend d'un étage dans l'arbre, on finit forcément par arriver sur un arbre vide et donc au cas de base.

c) Indiquer le symbole de comparaison manquant dans le test à la **ligne 22** de la méthode `insere` pour que l'arbre binaire de recherche réponde bien à la définition de l'encadré "**Rappel**".

Solution : Il faut mettre `elif self.racine.indice > nouveau_noeud.indice:`

d) On considère le robot dont la liste des tâches est représentée par l'arbre de la **figure 1**. Ce robot reçoit, successivement et dans l'ordre, des tâches d'indice de priorité 11, 5, 16 et 7, sans avoir accompli la moindre tâche entretemps. Recopier et compléter l'arbre de la **figure 1** après l'insertion de ces nouvelles tâches.

Solution : On obtient :



4) Avant d'insérer une nouvelle tâche dans l'arbre binaire de recherche, il faut s'assurer que son indice de priorité n'est pas déjà présent.

Écrire une méthode `est_present` de la classe `ABR` qui répond à la description :

```
    def est_present(self, indice_recherche) :
        """renvoie True si l'indice de priorité indice_recherche (int) passé
           en paramètre est déjà l'indice d'un noeud de l'arbre, False sinon"""
```

Solution :

```
    def est_present(self, indice_recherche) :
        if self.est_vide():
            return False
        elif indice_recherche < self.racine.indice:
            return self.racine.gauche.insere(indice_recherche)
        elif indice_recherche > self.racine.indice:
            return self.racine.droite.insere(indice_recherche)
        else:
            return True
```

5) Comme le robot doit toujours traiter la tâche dont l'indice de priorité est le plus petit, on envisage un parcours infixe de l'arbre binaire de recherche.

a) Donner l'ordre des indices de priorité obtenus à l'aide d'un parcours infixe de l'arbre binaire de recherche de la **figure 1**.

Solution : On obtiendra 6, 8, 10, 12, 13 et 14.

b) Expliquer comment exploiter ce parcours pour déterminer l'ordre dans lequel exécuter les tâches.

Solution : Les valeurs sont parcourus dans l'ordre croissant, donc celui dans lequel elles vont être exécutées.

6) Afin de ne pas parcourir tout l'arbre, il est plus efficace de rechercher la tâche du nœud situé le plus à gauche de l'arbre binaire de recherche : il correspond à la tâche prioritaire. Recopier et compléter la méthode récursive `tache_prioritaire` de la classe `ABR` :

```
def tache_prioritaire(self):
    """renvoie la tâche du noeud situé le plus
       à gauche de l'ABR supposé non vide"""
    if self.racine.____.est_vide(): #pas de noeud plus à gauche
        return self.racine.____
    else:
        return self.racine.gauche.____()
```

Solution :

```
def tache_prioritaire(self):
    if self.racine.gauche.est_vide(): #pas de noeud plus à gauche
        return self.racine.tache
    else:
        return self.racine.gauche.tache_prioritaire()
```

7) Une fois la tâche prioritaire effectuée, il est nécessaire de supprimer le nœud correspondant pour que le robot passe à la tâche suivante :

- Si le nœud correspondant à la tâche prioritaire est une feuille, alors il est simplement supprimé de l'arbre (cette feuille devient un arbre vide)
- Si le nœud correspondant à la tâche prioritaire a un sous-arbre droit non vide, alors ce sous-arbre droit remplace le nœud prioritaire qui est alors écrasé, même s'il s'agit de la racine.

Dessiner alors, pour chaque étape, l'arbre binaire de recherche (seuls les indices de priorités seront représentés) obtenu pour un robot, initialement sans tâche, et qui a, successivement dans l'ordre :

- étape 1 : reçu une tâche d'indice de priorité 14 à accomplir
- étape 2 : reçu une tâche d'indice de priorité 11 à accomplir
- étape 3 : reçu une tâche d'indice de priorité 8 à accomplir
- étape 4 : reçu une tâche d'indice de priorité 18 à accomplir
- étape 5 : reçu une tâche d'indice de priorité 15 à accomplir
- étape 6 : accompli sa tâche prioritaire
- étape 7 : accompli sa tâche prioritaire
- étape 8 : accompli sa tâche prioritaire
- étape 9 : reçu une tâche d'indice de priorité 20 à accomplir
- étape 10 : accompli sa tâche prioritaire

Solution :

1) 14

2) 14 — 11

3) 14 — 11 — 8

4) 14 — 11, 18 — 8

5) 14 — 11, 18 — 8, 15

6) 14 — 11, 18 — 15

7) 14 — 18 — 15

8) 18 — 15

9) 18 — 15, 20

10) 18 — 20

Exercice 3 : (5pt) *L'exercice porte sur l'architecture matérielle, les réseaux et les systèmes d'exploitation.*

Nous allons étudier les communications entre Bob et Alice. Ils communiquent au travers du réseau ci-dessous dont le protocole de routage est le protocole OSPF qui minimise le coût des communications.

Une adresse IPv4 est composée de quatre octets soit 32 bits. Une adresse de sous-réseau avec la notation /n signifie que les n premiers bits de l'adresse correspondent à la partie "réseau" et les suivants à la partie "machine".

L'adresse dont tous les bits de la partie "machine" sont à 0 est appelée adresse du réseau.

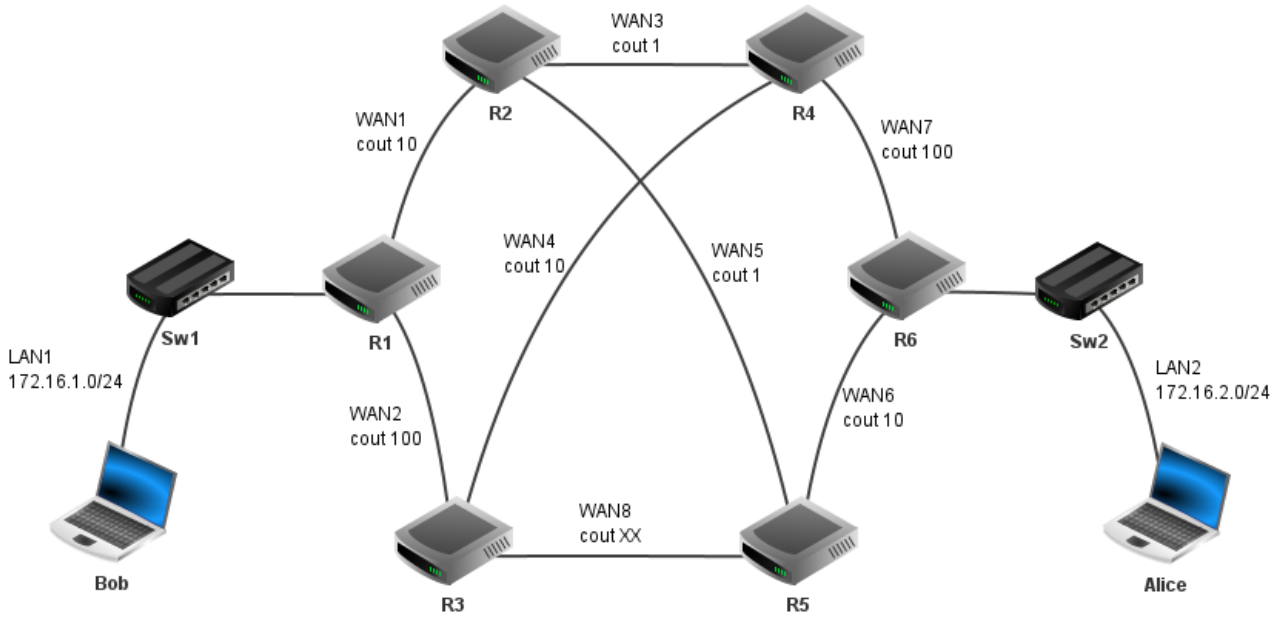
L'adresse dont tous les bits de la partie "machine" sont à 1 est appelée adresse de diffusion.

Ces adresses sont réservées et ne peuvent pas être attribuées à des machines.

Le choix des routes empruntées par les paquets IP est uniquement basé sur le protocole OSPF. On prendra comme débit maximal de référence 10 000 Mbit/s.

Le coût est alors calculé de la façon suivante :

$$\text{coût} = \frac{\text{débit maximal de référence}}{\text{débit du réseau concerné}}$$



LAN : réseau local ; WAN : réseau étendu ; R : routeur ; Sw : Switch

1) La configuration IP partielle ci-dessous a été affichée sur l'un des ordinateurs :

```
IP hôte : 172.16.2.3      IP passerelle : 172.16.2.253
```

Indiquer en justifiant si cette configuration appartient à l'ordinateur de Bob ou d'Alice.

Solution : Puisque l'IP commence par `172.16.2`, l'ordinateur est sur le réseau LAN2. C'est donc l'ordinateur d'Alice.

2) Le réseau WAN8 a un débit de 1 000 Mbit/s. Calculer le coût correspondant.
Solution : On calcule :

$$\text{coût} = \frac{10\,000}{1\,000} = 10$$

3) On donne les tables de routage des routeurs R1 à R5, dans lesquelles Pass. désigne la passerelle (qui correspond au routeur suivant).
Écrire sur votre copie la table de routage du routeur R6.

Routeur R1		
Destination	Pass.	Coût
LAN1	-	-
LAN2	R2	21
WAN1	-	-
WAN2	-	-
WAN3	R2	10
WAN4	R2	11
WAN5	R2	10
WAN6	R2	11
WAN7	R2	11
WAN8	R2	11

Routeur R2		
Destination	Pass.	Coût
LAN1	R1	10
LAN2	R5	11
WAN1	-	-
WAN2	R1	10
WAN3	-	-
WAN4	R4	1
WAN5	-	-
WAN6	R5	1
WAN7	R4	1
WAN8	R5	1

Routeur R3		
Destination	Pass.	Coût
LAN1	R4	21
LAN2	R5	20
WAN1	R4	11
WAN2	-	-
WAN3	R4	10
WAN4	-	-
WAN5	R5	10
WAN6	R5	10
WAN7	R4	10
WAN8	-	-

4) Bob envoie un message à Alice.
Énumérer dans l'ordre tous les routeurs par lesquels transitera ce message.

5) Un routeur tombe en panne, le nouveau coût pour la route entre Bob et Alice est de 111. Déterminer le routeur en panne. Justifier.

Routeur R4		
Destination	Pass.	Coût
LAN1	R2	11
LAN2	R2	12
WAN1	R2	1
WAN2	R3	10
WAN3	-	-
WAN4	-	-
WAN5	R2	1
WAN6	R2	2
WAN7	-	-
WAN8	R2	2

Routeur R5		
Destination	Pass.	Coût
LAN1	R2	11
LAN2	R6	10
WAN1	R2	1
WAN2	R3	10
WAN3	R2	1
WAN4	R2	2
WAN5	-	-
WAN6	-	-
WAN7	R2	2
WAN8	-	-

Solution :

4) Le message va passer par R1, R2, R5 et R6.

5) Pour changer de route, il faut qu'un des 4 routeurs empruntés soit en panne. Cela ne peut pas être R1 et R6 sinon la communication est impossible. Si R2 est en panne, la route est alors R1, R3, R5 et R6. Le coût est de 120. C'est donc R5 qui est en panne. La route a un coût de 111 et passe par R1, R2, R4 et R6.

Routeur R6		
Destination	Pass.	Coût
LAN1	R5	21
LAN2	-	-
WAN1	R5	11
WAN2	R5	21
WAN3	R5	11
WAN4	R5	12
WAN5	R5	10
WAN6	-	-
WAN7	-	-
WAN8	R5	10