Bac Général Maths + NSI (Mathématiques et Numérique et sciences informatiques)

SUJET D'EXAMEN (SESSION OFFICIELLE)

Numérique et sciences informatiques - Session 2023

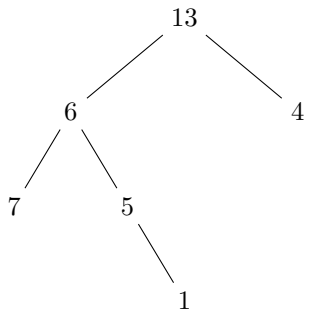
Sujet de l'examen national tel que publié pour la session indiquée. Source : sujets nationaux publiés par le ministère chargé de l'examen et les services académiques. Document mis à disposition par FormaV pour la révision.

Toutes les annales et leurs corrigés : https://bac-maths-nsi.fr/annales-sujets-et-corriges

Polynésie - mars 2023 - sujet 2

Exercice 1 (Arbres binaires de recherche, POO et récursivité - 4 points)

1. On considère l'arbre ci-dessous :



   (a) Justifier que cet arbre est un arbre binaire.

   (b) Indiquer si l'arbre ci-dessus est un arbre binaire de recherche (ABR). Justifier votre réponse.

2. On considère la classe `Noeud`, nous permettant de définir les arbres binaires, définie de la manière suivante en Python :

```
1  class Noeud:
2      def __init__(self, g, v, d):
3          """crée un noeud d'un arbre binaire"""
4          self.gauche = g
5          self.valeur = v
6          self.droit = d
```

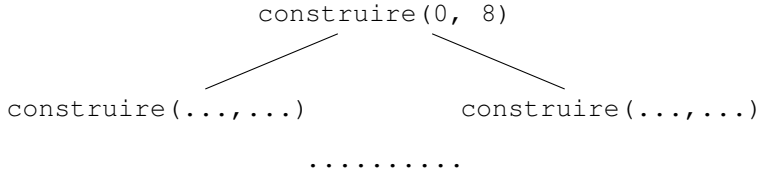
On considère également la fonction `construire` ci-dessous qui prend en paramètre deux entiers `mini` et `maxi` et qui renvoie un arbre.

```
1  def construire(mini, maxi):
2      """mini, maxi : entiers respectant mini <= maxi"""
3      assert isinstance(mini, int) and isinstance(..., ....) and ...
4      if maxi - mini == 1 or maxi - mini == 0:
5          return Noeud(None, mini, None)
6      elif maxi - mini == 2:
7          return Noeud(None, (mini+maxi)//2, None)
8      else:
9          sag = construire(mini, (mini/maxi)//2)
10         sad = construire((mini+maxi)//2, maxi)
11         return Noeud(sag, (mini+maxi)//2, sad)
```

La fonction `isinstance(obj, t)` renvoie `True` si l'objet `obj` est du type `t`, sinon `False`.

   (a) Recopier et compléter sur votre copie la ligne 3 de l'assertion de la fonction `construire` de manière à vérifier les conditions sur les paramètres `mini` et `maxi`.

   (b) On exécute l'instruction `construire(0, 8)`. Déterminer quels sont les différents appels récursifs de la fonction construire exécutés. On représentera ces appels sous forme d'une arborescence, par exemple :



(c) Dessiner l'arbre renvoyé par l'instruction `construire(0, 8)`.

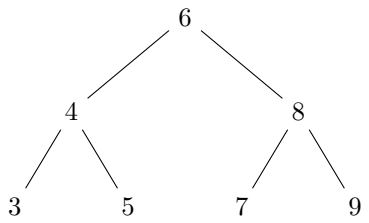
(d) Dessiner l'arbre renvoyé par l'instruction `construire(0, 3)`.

(e) Donner le résultat d'un parcours infixe sur l'arbre obtenu à la question 2.c.
Expliquer pourquoi ce parcours permet d'affirmer que l'arbre est un arbre binaire de recherche.

(f) La fonction récursive `maximum` ci-dessous prend en paramètre un arbre binaire de recherche `abr` et renvoie la valeur maximale de ses nœuds. Recopier et compléter les lignes 5 et 7 de cette fonction.

```
1 def maximum(abr):
2     if abr is None:
3         return None
4     elif abr.droit is None:
5         return ..................
6     else:
7         return ..................
```

3. On donne l'arbre binaire de recherche `abr_7_noeuds` suivant :



On donne également ci-dessous la fonction `mystere` qui prend en paramètres un arbre binaire de recherche `abr`, un entier `x` et une liste `liste`.

```
1  def mystere(abr, x, liste):
2      """
3      abr -- arbre binaire de recherche
4      x -- int
5      liste -- list
6      Renvoie -- list"""
7      if abr is None:
8          return []
9      else:
10         liste.append(abr.valeur)
11         if x == abr.valeur:
12             return liste
13         elif x < abr.valeur:
14             return mystere(abr.gauche, x, liste)
15         else:
16             return mystere(abr.droit, x, liste)
```

(a) Donner les résultats obtenus lorsque l'on exécute les instructions `mystere(abr_7_noeuds, 5, [])`, puis `mystere(abr_7_noeuds, 6, [])` puis `mystere(abr_7_noeuds, 2, [])`.

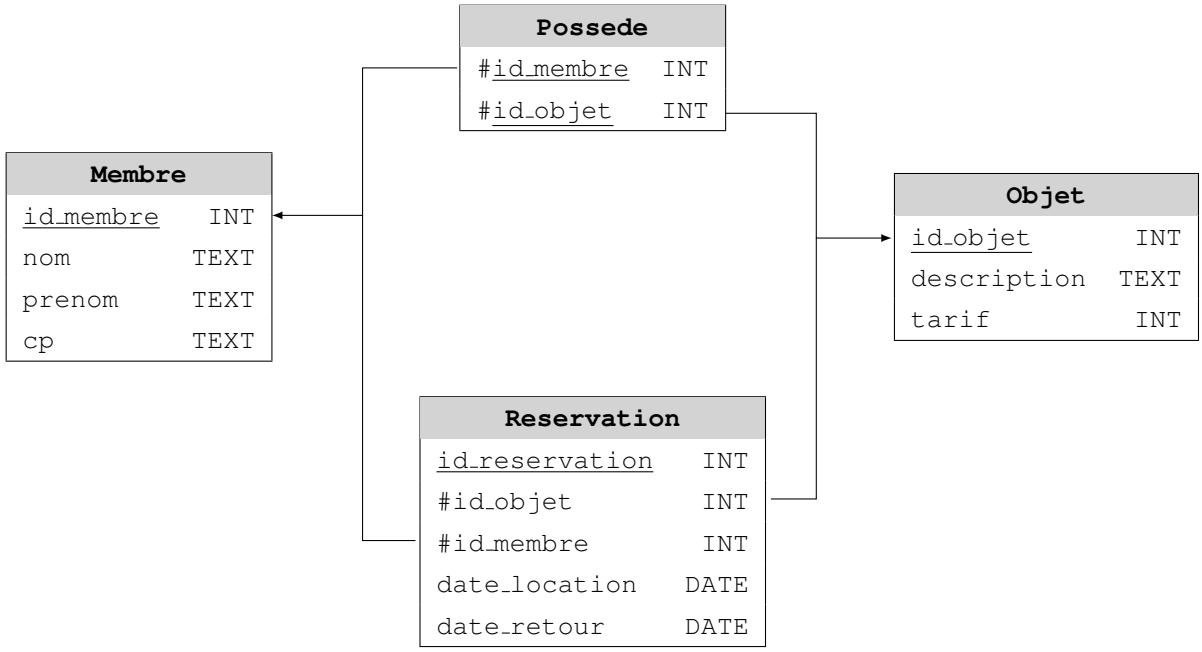
(b) Décrire quel peut être le rôle de la fonction `mystere`.

Exercice 2 (Bases de données et SQL - 4 points)
L'énoncé de cet exercice peut utiliser les mots du langage SQL suivants : `SELECT`, `FROM`, `WHERE`, `JOIN ON`, `INSERT INTO`, `VALUES`, `UPDATE`, `SET`, `DELETE`, `COUNT`, `AND`, `OR`.

Un site permet à ses membres de proposer à la location du matériel et de louer du matériel. Ceci permet de mutualiser du matériel entre membres et au propriétaire de rentabiliser cet achat. Le temps d'utilisation du matériel s'en trouve ainsi augmenté et le nombre d'appareils diminué.

Le modèle relationnel est donné par le schéma ci-dessous.

- La table `Membre` contient les informations de chaque utilisateur du site (nom, prénom et code postal).
- La table `Objet` décrit le type d'objet à la location ainsi que son tarif de location journalier.
- La table `Reservation` répertorie toutes les réservation effectuées par les membres du site avec notamment leur date de début et de fin de location.
- La table `Possede` permet de lier les tables `Membre` et `Objet`.



Les clés primaires sont soulignés et les clés étrangères sont précédées du symbole `#`. Les flèches indiquent les correspondances entre les clés étrangères et les clés primaires.

On donne ci-dessous le contenu de ces tables à un instant donné :

Membre

id_membre	nom	prenom	cp
1	"Ali"	"Mohamed"	"69110"
2	"Alonso"	"Fernando"	"69005"
3	"Dupont"	"Antoine"	"69003"
4	"Ferrand"	"Pauline"	"69160"
5	"Kane"	Harry"	"69003"

Possede

id_membre	id_objet
1	4
1	6
2	4
3	3
3	5
4	1
4	2

Objet

id_objet	description	tarif
1	"Nettoyeur haute pression"	20
2	"Taille-haie"	15
3	"Perforatrice"	15
4	"Appareil à raclette"	10
5	"Scie circulaire"	15
6	"Appareil à gauffre"	10

Reservation

id_reservation	id_objet	id_membre	date_location	date_retour
1	4	5	2022-02-18	2022-02-19
2	1	2	2022-05-05	2022-05-06
3	3	1	2022-07-10	2022-07-12
4	3	1	2022-08-12	2022-08-14
5	2	2	2022-10-20	2022-10-22
6	2	2	2022-10-20	2022-10-22

1. Dans cette partie, on ne demande pas de requête SQL. En étudiant le contenu des tables ci-dessus :
   (a) Indiquer quels sont les prénoms et noms du ou des membres du site qui proposent la location d'un appareil à raclette.
   (b) Donner le prénom et le nom du membre qui ne propose pas d'objet à la location.
2. (a) Donner le résultat de la requête suivante :

```
SELECT nom, prenom FROM Membre WHERE cp = "69003" ;
```

   (b) Ecrire une requête permettant de connaître le tarif de location d'une scie circulaire.
   (c) Ecrire une requête permettant de modifier le tarif de location d'un nettoyeur à haute pression pour le passer à 15 € par jour au lieu de 20 € par jour.
   (d) Ecrire une requête SQL permettant d'ajouter Wendie Renard habitant à Villeurbanne (code postal 69100) dans la table `Membre`, avec un `id_membre` de 6.
3. (a) Expliquer la limitation importante d'utilisation du service offert par le site si l'on utilisait le couple de clés étrangères `(id_objet, id_membre)` en tant que clé primaire de la relation `Reservation`.
   (b) Mohamed Ali décide de ne plus être membre du site. Il faut donc le supprimer de la table `Membre` à l'aide de la requête :

```
DELETE FROM Membre WHERE nom = "Ali" AND prenom = "Mohamed" ;
```

   Expliquer pourquoi cette requête produit une erreur.
   (c) Proposer une suite de requêtes utilisant le mot clé `DELETE`, précédant la requête ci-dessus pour supprimer correctement Mohamed Ali, dont l'`id_membre` est 1, de la base de donnée.
   Rappel : la relation `Objet` décrit le type d'objet à la location. Il n'y a pas d'objet à supprimer dans cette table lors du départ d'un membre.
4. Dans cette partie, les requêtes utilisent des jointures entre tables. On supposera donc les numéros `id_membre` et `id_objet` non connus.
   (a) Ecrire une requête permettant de compter le nombre de réservations réalisées par Fernando Alonso.
   (b) Ecrire une requête permettant de connaître les noms et prénoms des membres possédant un appareil à raclette.

Exercice 3 (Architecture matérielle, processus et ordonnancement - 4 points)

1. Avec la commande `ps -aef` on obtient l'affichage suivant :

```
 PID  PPID   C  STIME  TTY        TIME  CMD

8600     2   0  17:38  ?      00:00:00  [kworker/u2 :0-fl]
8859     2   0  17:40  ?      00:00:00  [kworker/0 :1-eve]
8866     2   0  17:40  ?      00:00:00  [kworker/0 :10-ev]
8867     2   0  17:40  ?      00:00:00  [kworker/0 :11-ev]
8887  6217   0  17:40  pts/0  00:00:00  bash
9562     2   0  17:45  ?      00:00:00  [kworker/u2 :1-ev]
9594     2   0  17:45  ?      00:00:00  [kworker/0 :0-eve]
9617  8887  21  17:46  pts/0  00:00:06  /usr/lib/firefox/firefox
9657  9617  17  17:46  pts/0  00:00:04  /usr/lib/firefox/firefox - contentproc -childID
9697  9617   4  17:46  pts/0  00:00:01  /usr/lib/firefox/firefox - contentproc -childID
9750  9617   3  17:46  pts/0  00:00:00  /usr/lib/firefox/firefox - contentproc -childID
9794  9617  11  17:46  pts/0  00:00:00  /usr/lib/firefox/firefox - contentproc -childID
9795  9794   0  17:46  pts/0  00:00:00  /usr/lib/firefox/firefox
9802  7441   0  17:46  pts/2  00:00:00  ps -aef
```

On rappelle que :

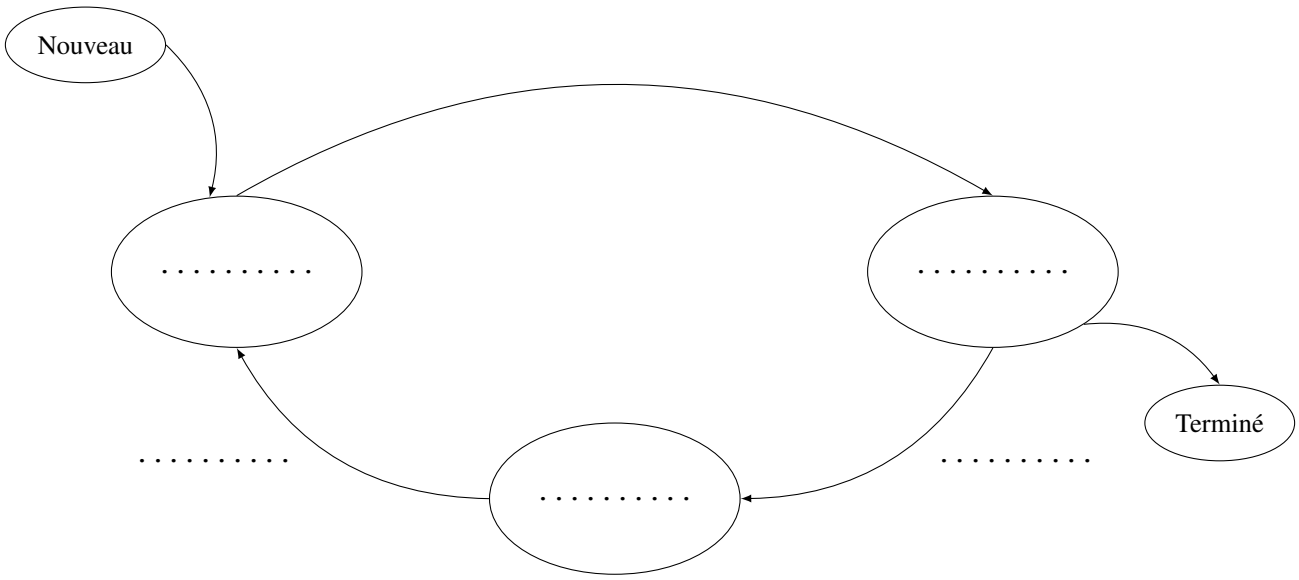
- ⋆ PID = Identifiant d'un processus (Process Identification)
- ⋆ PPID = Identifiant du processus parent d'un processus (Parent Process Identification)

(a) Donner sous forme d'un arbre de PID la hiérarchie des processus liés à `firefox`.

(b) Indiquer la commande qui a lancé le premier processus de `firefox`.

(c) La commande `kill` permet de supprimer un processus à l'aide de son PID (par exemple `kill 8600`). Indiquer la commande qui permettra de supprimer tous les processus liés à `firefox` et uniquement cela.

2. (a) Recopier et compléter le schéma ci-dessous avec les termes suivants concernant l'ordonnancement des processus : *Elu, En attente, Prêt, Blocage, Déblocage, Mise en exécution*.



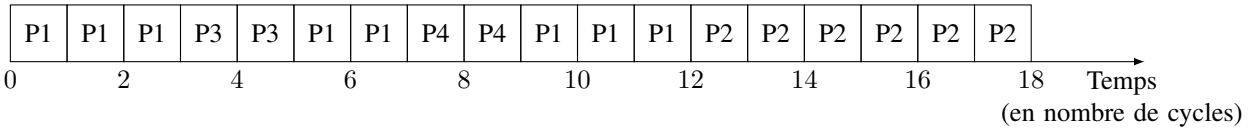
On donne dans le tableau ci-dessous quatre processus qui doivent être exécutés par un processeur. Chaque processus a un instant d'arrivé et une durée, donnés en nombre de cycles du processeur.

Processus	**P1**	**P2**	**P3**	**P4**
Instant d'arrivée	0	2	3	7
Durée	8	6	2	2

Les processus sont placés dans une file d'attente en fonction de leur instant d'arrivée. On se propose de les ordonnancer avec la méthode suivante :

- ⋆ parmi les processus présents en liste d'attente, l'ordonnanceur choisit celui dont la durée restante est la plus courte ;
- ⋆ le processeur exécute un cycle de ce processus puis l'ordonnanceur désigne de nouveau le processus dont la durée restante est la plus courte ;
- ⋆ en cas d'égalité de temps restant entre plusieurs processus, celui choisi sera celui dont l'instant d'arrivée est le plus ancien ;
- ⋆ tout ceci jusqu'à épuisement des processus en liste d'attente.

On donne en exemple ci-dessous, l'ordonnancement des quatre processus de l'exemple précédent suivant l'algorithme ci-dessus :

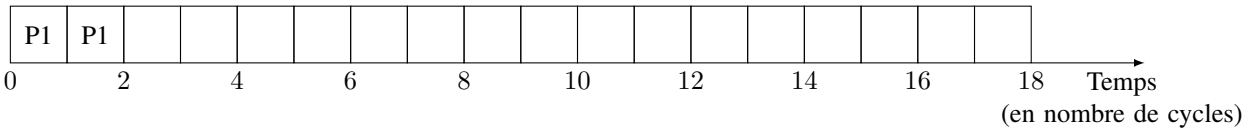


On définit le temps d'exécution d'un processus comme la différence entre son instant de terminaison et son instant d'arrivée.

(b) Calculer la moyenne des temps d'exécution des quatre processus.

On se propose de modifier l'ordonnancement des processus. L'algorithme reste identique à celui présenté précédemment mais au lieu d'exécuter un seul cycle, le processeur exécutera à chaque fois deux cycles du processus choisi. En cas d'égalité de temps restant, l'ordonnanceur départagera toujours en fonction de l'instant d'arrivée.

(c) Recopier et compléter le schéma ci-dessous donnant le nouvel ordonnancement des quatre processus.



(d) Calculer la nouvelle moyenne des temps d'exécution des quatre processus et indiquer si cet ordonnancement est plus performant que le précédent.

3. On se propose de programmer l'algorithme du premier ordonnanceur. Chaque processus sera représenté par une liste comportant autant d'éléments que de durées (en nombre de cycles). Pour simuler la date de création de chaque processus, on ajoutera en fin de liste de chaque processus autant de chaînes de caractères vides que la valeur de leur date de création.

```
p1 = ['1.8', '1.7', '1.6', '1.5', '1.4', '1.3', '1.2', '1.1']
p2 = ['2.6', '2.5', '2.4', '2.3', '2.2', '2.1', '', '']
p3 = ['3.2', '3.1', '', '', '']
p4 = ['4.2', '4.1', '', '', '', '', '', '', '']
```

La fonction `choix_processus` est chargée de sélectionner le processus dont le temps restant d'exécution est le plus court parmi les processus en liste d'attente.

(a) Recopier sans les commentaires et compléter la fonction `choix_processus` ci-dessous. Le code peut contenir plusieurs lignes.

```
def choix_processus(liste_attente):
    """Renvoie l'indice du processus le plus court parmi
    ceux présents en liste d'attente liste_attente"""
    if liste_attente != []:
        mini = len(liste_attente[0])
        indice = 0
        .....
        return indice
```

Une fonction `scrutation` (non étudiée) est chargée de parcourir la liste `liste_proc` de tous les processus et de renvoyer la liste d'attente des processus en fonction de leur arrivée. À chaque exécution de `scrutation`, les processus présents (sans chaînes de caractères vides en fin de liste) sont ajoutés à la liste d'attente. La fonction supprime pour les autres un élément de chaîne de caractères vides.

(b) Recopier et compléter les différentes instructions de la fonction `ordonnancement` pour réaliser le fonctionnement désiré.

```
def ordonnancement(liste_proc):
    """Exécute l'algorithme d'ordonnancement
    liste_proc -- liste des processus
    Renvoie la liste d'exécution des processus"""
    execution = []
    attente = scrutation(liste_proc, [])
    while attente != []:
        indice = choix_processus(attente)
        ..... # A FAIRE (plusieurs lignes de code)
        attente = scrutation(liste_proc, attente)
    return execution
```