BACCALAURÉAT BLANC
SESSION 2024

Épreuve de l'enseignement de spécialité
NUMÉRIQUE et SCIENCES INFORMATIQUES

Partie écrite
Classe Terminale de la voie générale
DURÉE DE L'ÉPREUVE : 3 heures 30 min

Janvier 2024

Le sujet comporte 13 pages numérotées de 1 à 13.

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

L'usage de la calculatrice n'est pas autorisé

Le candidat doit traiter tous les exercices

Exercice 1 (8 points)

Cet exercice porte sur la notion de liste, de tris et les algorithmes de recherche.

Partie A : Équipes de football

Pour faciliter la gestion d'un club de sport, on souhaite créer un algorithme qui va réaliser la gestion des compétitions entre équipes.

Une équipe est un ensemble de joueurs qui ont tous un *nom*, un *nombre de but* marqué lors des matchs et un identifiant unique.
Il est possible qu'un nom ou un nombre de but soit présent plusieurs fois dans une équipe mais pas l'identifiant.
Nos équipes seront stockées dans des listes de listes contenant dans l'ordre un entier pour l'identifiant, un entier pour le nombre de but ainsi qu'une chaîne de caractères pour le nom.

Un exemple d'équipe :

equipe_1 = [[1, 5, «Paul»], [7,0, « Didier »], [2, 3, « Marc »]]

Ici l'équipe est composée de Paul identifié par le numéro 1 et qui a marqué 5 buts, par Didier numéro 7 qui n'a pas marqué et de Marc qui a le numéro 3 pour 2 but marqués.

1. Écrire une fonction *nom_membres_equipe(equipe)* qui prends en paramètre une liste de liste et qui renvoie une liste avec l'ensemble des noms des joueurs d'une équipe.
2. Écrire une fonction *nombre_buts_equipe(equipe)* qui prends en paramètre une liste de liste et qui renvoie un entier correspondant au nombre de but marqué au total par l'équipe passé en paramètre de la fonction.

Une équipe est valide seulement si elle a au moins un membre et que chacun des membres à bien un identifiant unique.

3. Corriger la fonction *validation_equipe(equipe)* qui vérifie les conditions énumérées ci-dessus.

```
def validation(equipe):
    res = True

    for i in equipe:
        for j in equipe:
            if i == j and i[0] == j[1]:
                res = False

    return len(equipe)<0 or res
```

Partie B : Compétitions

Les équipes participent à des compétitions qui sont représentées par des listes d'équipe. Rappel les équipes sont définies par des listes de listes.
Imaginons d'autres équipes :

equipe_2 = [[42, 2, «Blaise»], [1, 1, « Pascal »], [3, 2, «Matuidi »]]

equipe_3 = [[88, 5, «Victor»], [7,0, « Hugo »], [2, 3, « Zidane »]]

equipe_4 = [[3, 78, «Paul»], [15 , 1 «Valéry»], [21, 68, « Marc »]]

La compétition représente l'ordre des matchs. Le nombre de buts de chaque joueur correspond au nombre de buts totaux marqués lors de la compétition.

competition = [[equipe_1, equipe_2, equipe_3, equipe_4], [equipe_1, equipe_4] , [equipe_1]]

La 1er sous-liste correspondant à l'ensemble des équipes au départ, la seconde aux vainqueuses des 2 premiers match et la dernière au vainqueuses de la compétition.

4. Écrire equipe_ayant_le_plus_marque(*competition*) prenant une liste de liste en paramètre et qui renvoie l'équipe ayant le plus marqué.
5. L'équipe ayant le plus marquée est-elle forcément sûre de gagner le tournoi ? Démontrer votre réponse avec un exemple.
6. Écrire *equipe_vainqueuses(competition)* prenant une liste de liste en paramètre qui renvoie l'équipe vainqueuses. Votre fonction doit avoir une complexité constante.
7. Compléter la fonction *equipe_perdantes(competition)* prenant une liste d'équipes et qui renvoie une liste d'équipes ayant perdu.

```
def equipe_perdantes(competition):
    res = []

    for i in competition[0]:
        if ... != ... :
            res.append(...)
    return res
```

Nous avons ici une structure assez lourde, la liste de liste qui nous oblige à utiliser des indices et pouvant être sources d'erreur.

Cela n'est sûrement pas la meilleure structure de données possible.

8. Il existe une structure permettant de récupérer en temps constant des valeurs via un identifiant unique , donner son nom.
9. Créer une variable *mon_equipe* qui reprend les valeurs de *equipe_1* sous la forme de la structure définie dans la question précedente.

Partie C : Tris des équipes

Nous allons dans cette partie garder la structure de liste de liste d'équipe.

Imaginons une équipe plus grande :

equipe_5 = *[[1, 2, «Boris»], [7,0, «Vian»], [2, 3, «Claude»] , [28, 3, «François»], [12, 3, «Patrick»], [38, 3, «Sébastien»], [27, 3, «Thierry»], [3, 3, «Henry»], [52, 3, «Michael»], [42, 3, «Jordan»]]*

10. Écrire est_present(equipe, id) prenant une équipe représentée par une liste de liste et un entier pour l'identifiant. Cette fonction renvoie vraie seulement si il y a dans la liste un joueur avec un identifiant équivalent au paramètre *id.*
11. Donner le meilleur et le pire des cas dans la recherche linéaire d'un élément dans cette liste.
12. Implémenter en python un algorithme vu en cours permettant de trier les équipes par ordre croissant d'identifiant.
13. Quel est le nom de ce tri ?
14. Créons une equipe_6 qui est une copie de equipe_5, on applique la fonction de tri sur equipe_6. L'appel de est_present sur equipe_6 est-il plus efficace ? Justifier.
15. La fonction mystère suivante ne peut fonctionner qu'avec une équipe triée dans l'ordre croissant. Que fait-elle ?

```
def fonction_mystere(equipe, v):
    a = 0
    b = len(equipe) - 1
    while a <= b:
        m = (a + b) // 2
        if equipe[m][0] == v:
            return True
        elif equipe[m][0] < v:
            a = m + 1
        else:
            b = m - 1
    return False
```

16. Écrire cet algorithme de manière récursive.

Exercice 2 : Bases de Données Relationnelles /6

Cet exercice porte sur les bases de données relationnelles et les requêtes SQL.
Un journaliste souhaite renouveller sa base de données qui permet de stocker et traiter des informations sur les jeux-vidéos dont il a fait des articles.
Il désire stocker les informations suivantes :

- L'identifiant du jeu-video qui est unique (`id`) (entier).
- Le nom du jeu (`nom`) (chaîne de caractères).
- Le nom du studio qui a réalisé le jeu (`studio`) (chaîne de caractères).
- L'année de sortie du jeu (`annee_sortie`) (entier).
- Une note sur 10 qui correspond à la moyenne des notes des critiques (`note_critique`) (entier).
  On utilise une base de données relationnelles. Les commandes ont déjà été exécutées afin de créer une table `jeu_video` .
  Cette relation `jeu_video` contient toutes les données sur les jeux-vidéos. On obtient donc la table suivante ***(non exhaustive, un extrait)***:

		jeu_video		
id	***nom***	***studio***	***annee_sortie***	***note_critique***
12	**The Witcher 3 : Wild Hunt**	**CD Projekt**	**2015**	**9**
2	**Red Dead Redemption 2**	**Rockstar Games**	**2018**	**9**
32	**God Of War**	**Santa Monica Studio**	**2018**	**10**
29	**Cyberpunk 2077**	**CD Projekt**	**2020**	**9**
5	**Assassin's Creed Mirage**	**Ubisoft**	**2023**	**8**
6	**Fallout 4**	**Bethesda Games Studios**	**2015**	**8**
15	**The Legend Of Zelda : Breath Of The Wild**	**Nintendo**	**2017**	**10**
8	**Overwatch**	**Blizzard Entertainment**	**2016**	**6**
19	**SpiderMan : Miles Morales**	**Insomniac Games**	**2020**	**8**
10	**Horizon Zero Dawn**	**Guerrilla Games**	**2017**	**9**

1. Donner le schéma relationnel de la relation `jeu_video`.
2. Expliquer pourquoi l'attribut *studio* ne peut pas être choisi comme clef primaire.
3. Donner le résultat renvoyé par la requête SQL suivante :

```sql
SELECT nom
FROM jeux_video
WHERE studio = 'CD Projekt';
```

4. Écrire une requête SQL permettant d'obtenir les noms des jeux développés par Ubisoft après 2013.
5. Écrire une requête SQL permettant de modifier la note du jeu OverWatch en la passant de 6 à 4.
6. Écrire une requête SQL permettant d'ajouter le jeu *TitanFall*, qui a l'id *42*, développé par le studio *Respawn Entertainment*, sorti en *2014* et qui a reçu une note de *7*.
7. Écrire une requête SQL permettant de lister les jeux développés par *Nintendo* ayant eu une note inférieure à 6.
8. Écrire une requête SQL permettant de supprimer le jeu *Fallout 4* de la table `jeu_video`.

On souhaite proposer plus d'informations sur les studios qui développent les jeux-vidéos. Pour cela, on créée une deuxième table `studio` avec les attributs suivants :

- `id` de type entier
- `nom` de type chaîne de caractères
- `pays` de type chaîne de caractères
- `annee_creation` de type entier

	studio		
id	***nom***	***pays***	***annee_creation***
1	**CD Projekt**	**Pologne**	**1994**
2	**Rockstar Games**	**États-Unis**	**1998**
3	**Santa Monica Studio**	**États-Unis**	**1999**
4	**Ubisoft**	**France**	**1986**
5	**Nintendo**	**Japon**	**1889**
6	**Blizzard Entertainment**	**États-Unis**	**1991**
7	**Insomniac Games**	**États-Unis**	**1994**
8	**Guerrilla Games**	**Pays-Bas**	**2000**

La table `jeux_video` en est donc aussi modifiée.

		jeu_video		
id	*nom*	*id_studio*	*annee_sortie*	*note_critique*
12	**The Witcher 3 : Wild Hunt**	**1**	**2015**	**9**
2	**Red Dead Redemption 2**	**2**	**2018**	**9**
32	**God Of War**	**3**	**2018**	**10**
29	**Cyberpunk 2077**	**1**	**2020**	**9**
5	**Assassin's Creed Mirage**	**4**	**2023**	**8**
15	**The Legend Of Zelda : Breath Of The Wild**	**5**	**2017**	**10**
8	**Overwatch**	**6**	**2016**	**6**
19	**SpiderMan : Miles Morales**	**7**	**2020**	**8**
10	**Horizon Zero Dawn**	**8**	**2017**	**9**

9. Expliquer quelle est l'utilité d'utiliser deux tables (`jeux_video` et `studio`) au lieu d'avoir toutes les informations au sein d'une même table.
10. Expliquer quel est le rôle de l'attribut *id_studio* dans la table `jeux_video` .
11. Écrire une requête SQL permettant d'afficher les noms des jeux qui ont été développés par un studio français.

Le journaliste se demande si utiliser un fichier CSV et un fichier Python pour effectuer ses requêtes ne serait pas plus pratique.

Pour se faire, il créée une classe `jeu_video` .

```
class jeu_video:
    def __init__(self, nom, studio, annee_sortie,note_critique):
        '''
        Initialise un jeu vidéo à l'aide de son nom (str), son studio de développement (str),
        son année de sortie et la note donnée par les critiques ).
        '''
        assert int(note_critique) <= 10 and int(note_critique) >= 0
        self.nom = nom
        self.studio = studio
        self.annee_sortie = annee_sortie
        self.note_critique = note_critique

    def __str__(self):
        return f"{self.nom} est un jeu de {self.studio} sorti en {self.annee_sortie}
               et a eu une note critique de {self.note_critique}/10"
```

Pour organiser sa base de données, il pense à créer une classe `ludotheque`.
Cette ludothèque est modélisée par une liste de jeux-vidéos alimentée par un fichier CSV.

L'instruction *with open* à la ligne 8 permet d'ouvrir le fichier en lecture.
La méthode *DictReader* à la ligne 12 permet de lire le fichier CSV comme un dictionnaire.
La boucle de la ligne 13 à 18 ajoute chaque objet jeu dans la ludothèque pour la compléter.

```
1  class ludotheque:
2      def __init__(self, csv_file):
3          '''
4          Créée une liste de jeux-vidéos via l'ouverture d'un fichier CSV passé en paramètres
5          et la remplit pour chaque ligne lue du fichier.
6          '''
7          self.ludotheque = []
8          #Ouverture du fichier CSV
9          with open(csv_file, "r") as csv_file:
10             #Création d'un dictionnaire à l'aide des attributs et des enregistrements
11             #du fichier CSV.
12             dico_jeu = csv.DictReader(csv_file)
13             for ligne in dico_jeu:
14                 #Utilise le constructeur de jeu-vidéo pour créer une instance d'un jeu et
15                 #l'ajoute dans la ludothèque
16                 jeu = jeu_video(ligne["nom"], ligne["studio"],
17                                 ligne["annee_sortie"], ligne["note_critique"])
18                 self.ludotheque.append(jeu)
```

12. Que signifie CSV? Comment est-il construit?
13. Compléter la méthode de la classe `ludotheque` *jeux_sortis_avant* qui prend en paramètre une liste de jeux-vidéos et un entier qui représente l'année et renvoie les jeux sortis avant l'année demandée.

```
1      def jeux_sortis_avant(self, annee):
2          '''
3          Renvoie une liste de jeux-vidéos de type jeu_video sortie avant
4          l'année passée en paramètre (de type int).
5          '''
6          jeux = []
7          for jeu in ... :
8              if jeu.annee_sortie == ...:
9                  jeux.append(...)
10         return ...
```

14. En déduire une méthode de la classe `ludotheque` *jeux_du_studio* qui prend en paramètre une liste de jeux-vidéos et une chaine de caractère représentant le nom du studio et renvoie la liste des jeux-vidéos du studio.

Le journaliste voudrait créer une méthode de la classe `ludotheque` *ajouter_jeu* qui ajoute un jeu dans la ludothèque et dans son fichier CSV.
Cette méthode ajoute le je u passé en paramètre dans la ludothèque.
La méthode *open* à la ligne 9 permet d'ouvrir le document csv en écriture grâce à la méthode *DictWriter* du module `csv`.
La méthode *writerow* à la ligne 11 permet de rajouter dans le fichier CSV le dictionnaire passé en paramètre, le considérant comme un enregistrement.

```
1     def ajouter_jeu(self, jeu):
2     '''
3     Ajoute un jeu de type jeu_video à la ludothèque et ajoute son enregistrement
4     dans le fichier CSV servant à la construction de la base.
5     '''
6     self.ludotheque.append(jeu)
7     champs = ['nom', 'studio', 'annee_sortie', 'note_critique']
8     # Ajout du jeu sur une nouvelle ligne CSV
9     with open("jeux.csv", "a", newline='') as csv_file:
10        jeu_a_rajouter = csv.DictWriter(csv_file, fieldnames=champs)
11        jeu_a_rajouter.writerow({'nom': jeu.nom, 'studio': jeu.studio,
12                                 'annee_sortie': jeu.annee_sortie,
13                                 'note_critique': jeu.note_critique})
```

15. Quelle est la meilleure manière de réaliser sa base de données (Système de Gestion de Base de Données ou Fichier CSV et algorithme Python) ? Argumenter en se basant sur les propriétés ACID.

Exercice 3 (6 points) :

Cet exercice traite des structures de données et des algorithmes associés, notamment des files et des arbres binaires.

Les algorithmes de cet exercice devront être écrits en langage naturel.

Une file de priorité est un type abstrait contenant des éléments ayant chacun une priorité représentée par un nombre entier positif. On veut pouvoir effectuer les deux opérations suivantes de manière efficace :

- **`enfiler_p(x, f)`** qui ajoute un nouvel élément de priorité **`x`** dans la file **`f`**
- **`defiler_p(f)`** qui renvoie et supprime de la file **`f`** l'élément ayant la plus grande priorité

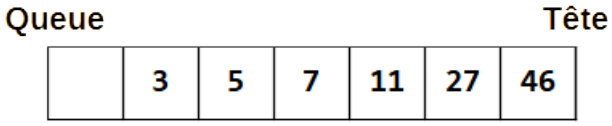
Les files de priorité sont donc très proches des piles et des files (les opérations **`enfiler_p`** et **`defiler_p`** sont semblables) mais au lieu de ressortir les éléments dans un ordre dépendant de l'ordre dans lequel ils ont été insérés, on veut les ressortir dans un ordre qui dépend de leur priorité.

1. Proposer une structure de donnée permettant d'implémenter à la fois la valeur et la priorité des éléments.

D'un point de vue algorithmique la valeur de l'élément n'a aucune importance, seule compte la valeur de sa priorité. Donc dans tout cet exercice, on ne considérera que la priorité des éléments, c'est-à-dire que l'on insérera directement des entiers dans la file. La fonction **`defiler_p(f)`** devra simplement renvoyer le plus grand entier contenu dans la file de priorités.

Partie A : file de priorité comme une file classique

On envisage la file de priorité comme une file triée, l'élément de tête ayant la valeur la plus élevée, les suivants de valeurs de plus en plus faibles. Ainsi, défiler la file permettra de prélever l'élément de plus haute priorité. Dans l'exemple ci-dessous, ce sera donc l'élément de valeur **`46`** qui sera défiler en premier.



1. Cette file de priorité est-elle du type LIFO, FIFO ou aucun des deux ? Justifier.

On dispose du type abstrait file classique muni des primitives :

- **`est_vide(f)`** qui renvoie Vrai si la file est vide
- **`enfiler(x, f)`** qui ajoute un nouvel élément **`x`** en queue de file **`f`**
- **`defiler(f)`** qui renvoie et supprime de la file **`f`** l'élément de tête

2. Justifier que l'opération **`defiler`** et **`defiler_p`** sont identiques.

Pour enfiler un élément de priorité **x** à sa place dans la file, on construit une nouvelle file **g** en défilant de manière pertinente la file **f** avant de pouvoir enfiler l'élément **x** dans la file **g**.

3. En utilisant la file exemple, dire jusqu'à quel élément doit-on défiler la file **f** dans la file **g** avant de pouvoir enfiler un élément de priorité **13** ?

4. Comment compléter la file **g** pour qu'elle contienne tous les éléments de la file **f** et le nouvel élément de priorité **13** à sa place dans la file ?

5. En déduire un algorithme pour la primitive **enfile_p** utilisant les 3 primitives des files classiques.

On considère une file de priorité contenant **N** éléments.

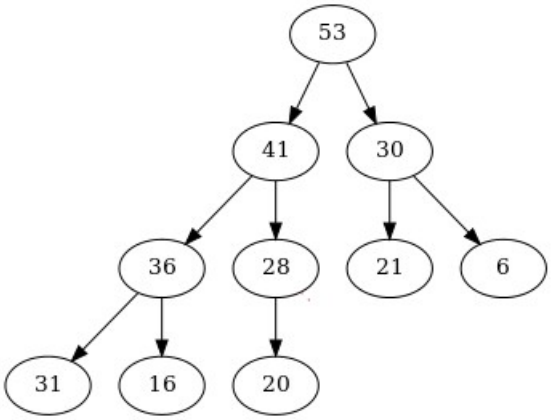
6. Quel est le pire des cas pour la primitive **enfiler(x, f)** ?

7. Que vaut la complexité en temps de cette opération dans ce cas ?

Partie B : file de priorité comme un tas binaire

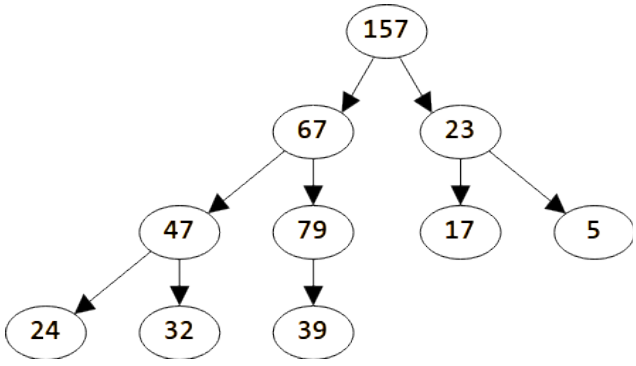
Un tas binaire est un type abstrait de type arbre binaire dont chaque nœud père a une étiquette de valeur supérieure à chacun de ses nœuds fils. Chacun des niveaux de l'arbre sont pleins à l'exception du dernier. Les nœuds du niveau le plus profonds sont le plus à gauche possible.

L'arbre ci-dessous, qui sera pris comme exemple dans tout l'exercice, est un tas.



Le nœud racine est considéré à une profondeur de **0**. L'arbre vide à une taille de **-1**.

1. L'arbre ci-contre n'est pas un tas. Pourquoi ? **Ne plus utiliser cet arbre pour la suite de l'exercice.**



2. Rappeler la définition d'un arbre binaire.

3. Dire si les propositions ci-dessous sont vraies ou fausses :

a. Un tas binaire est forcément strict
b. Un tas binaire est forcément complet
c. Un tas binaire est forcément parfait

4. Donner la racine et les feuilles du tas donné en exemple.

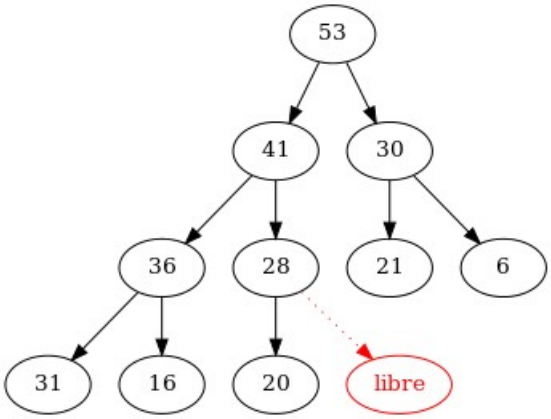
5. Quelle est sa hauteur ? Quelle est sa taille ? Quelle est la profondeur du nœud d'étiquette 28 ?

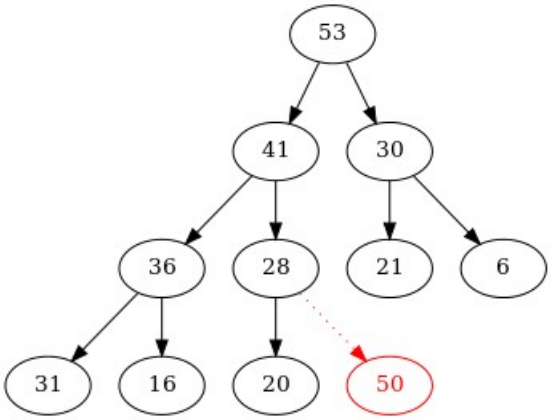
Le tas binaire étant pré-trié, il est utilisé pour faciliter les opérations de tri ou l'implémentation des files d'exécutions. Dans le deuxième cas qui nous intéresse pour cet exercice :

- **`enfiler_p(x, f)`** correspond à l'insertion d'un nœud de priorité **`x`** à sa place dans le tas **`f`**
- **`defiler_p(f)`** renvoie et supprime le nœud racine du tas **`f`**

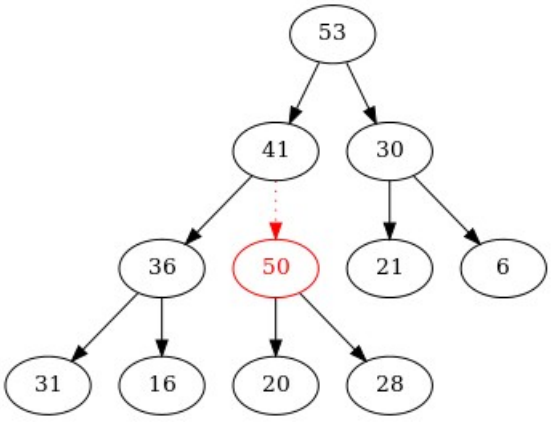
Que l'on applique l'une ou l'autre des primitives, on obtient à chaque fois un nouveau tas qui doit garder ses propriétés de tas.

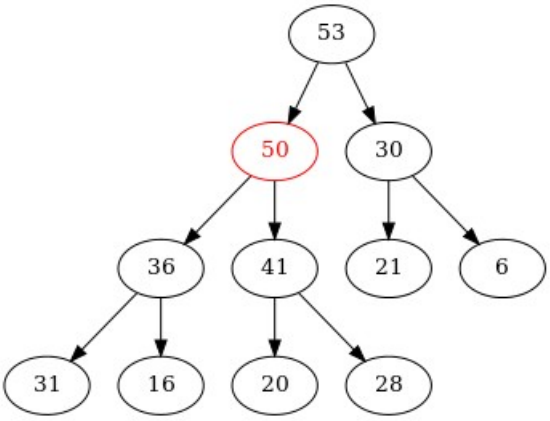
Par exemple, pour insérer le nœud de priorité **`50`**, on l'insère d'abord à la position libre la plus à gauche.





Il n'est pas à sa place. Il faut le faire remonter en comparant sa valeur à celle de son nœud père. Ici **`50`** est supérieur à **`28`** donc on permute les valeurs. On continue ainsi tant que la valeur du nœud fils est supérieure à celle du nœud père.





6. A partir de l'arbre dont on a inséré le nœud de valeur **`50`**, donner l'arbre obtenu en ajoutant en plus la valeur **`57`**.

On rappelle les encadrements de la taille **N** et de la hauteur **h** d'un arbre binaire :

$$\lfloor \log_2(N) \rfloor \leq h \leq N-1$$

$$h+1 \leq N \leq 2^{h+1}-1$$

7. Quelle est la hauteur d'un tas de **N** nœuds. Justifier.

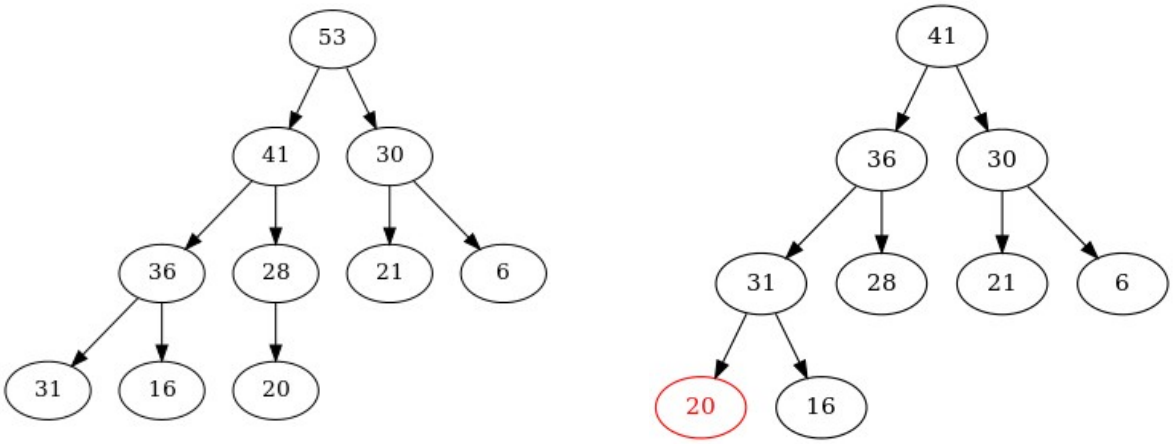
8. Quel est le pire des cas pour la primitive **`enfiler(x, f)`** ?

9. Expliquer pourquoi la complexité en temps de l'opération est logarithmique dans ce cas.

10. Implémenter le type abstrait file de priorité est-il plus efficace avec une file classique ou un tas ? Argumenter en calculant notamment les complexités en temps des deux approches pour une file de priorité contenant **172** éléments.

A faire en fin d'épreuve uniquement si vous avez le temps :

11. Donner un algorithme permettant d'implémenter la primitive **`defiler_p`** à partir d'un tas. Ci-dessous à gauche, le tas avant suppression de l'élément de plus haute priorité, à droite, le tas en fin d'algorithme.



Sources : Wikipedia