BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2022

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Lundi 31 janvier 2022

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 13 pages numérotées de 1/13 à 13 /13.

Le candidat traite au choix 5 exercices
parmi les 5 exercices proposés

(et oui, tous les exercices !)

Chaque exercice est noté sur 4 points

EXERCICE 1 (4 points)

Cet exercice porte sur les bases de données relationnelles et le langage SQL.

L'énoncé de cet exercice utilise les mots du langage SQL suivant :

`SELECT`, `FROM`, `WHERE`, `JOIN`, `INSERT INTO`, `VALUES`, `ORDER BY`

On rappelle qu'en SQL la clause `ORDER BY`, suivie d'un attribut, permet de classer les résultats par ordre croissant de l'attribut.
Dans un lycée, le parc informatique est constitué d'ordinateurs, d'imprimantes, de vidéoprojecteurs et de TNI (Tableau Numérique Interactif).
Tous les ordinateurs et toutes les imprimantes sont connectés au réseau de l'établissement.
Chaque salle de cours est identifiée par un numéro unique et contient :

- un ou plusieurs ordinateurs reliés au réseau de l'établissement ;
- aucun ou un seul vidéoprojecteur ;
- s'il y a un vidéoprojecteur, aucun ou un seul TNI ;
- une ou plusieurs imprimantes réseau.

Un ordinateur peut être connecté via le réseau à une ou plusieurs imprimantes (situées éventuellement dans une ou plusieurs autres salles) et à un vidéoprojecteur avec TNI ou non.
Les ordinateurs et les imprimantes possèdent un nom unique sur le réseau de l'établissement.
Les vidéoprojecteurs ne sont pas connectés au réseau, ils ne possèdent pas de nom unique. Ils sont donc identifiés par le numéro de la salle où ils sont installés.
Tous ces matériels sont gérés à l'aide d'une base de données relationnelle qui comprend 3 relations (ou tables) nommées : `Ordinateur`, `Videoprojecteur`, `Imprimante`.

On donne ci dessous le schéma relationnel de la relation `Ordinateur`, suivi d'un extrait de la relation `Ordinateur`. La clé primaire est soulignée.

`Ordinateur`(`nom_ordi` : String, `salle` : String, `marque_ordi` : String, `modele_ordi` : String, `annee` : Int, `video` : Boolean)

On distingue deux types d'ordinateurs :

- les ordinateurs multimédias pour les logiciels généraux avec un nom commençant par le groupe de lettres Gen.
- les ordinateurs techniques pour les logiciels qui demandent plus de ressources avec un nom commençant par le groupe de lettres Tech.

Ce groupe de lettres est suivi d'un tiret et du numéro unique de l'ordinateur pour chaque type.

Les 5 premières lignes de la relation `Ordinateur`

nom_ordi	salle	marque_ordi	modele_ordi	annee	video
Gen-24	012	HP	compaq pro 6300	2012	true
Tech-62	114	Lenovo	p300	2015	true
Gen-132	223	Dell	Inspiron Compact	2019	true
Gen-133	223	Dell	Inspiron Compact	2019	false
Gen-134	223	Dell	Inspiron Compact	2019	false

1. (a) À l'aide d'un système de gestion de base de données, on envoie au serveur la requête SQL suivante :

SELECT `salle`, `marque_ordi` FROM `Ordinateur` ;

Quel résultat produit cette requête sur l'extrait de la relation `Ordinateur` donné ci-dessus ?

(b) Quel résultat produit la requête suivante sur l'extrait de la relation `Ordinateur` donné ci-dessus ?

SELECT `nom_ordi`, `salle` FROM `Ordinateur` WHERE `video` = true ;

2. Écrire une requête SQL donnant tous les attributs des ordinateurs correspondant aux années supérieures ou égales à 2017 ordonnées par dates croissantes.
3. (a) Pour quelle raison l'attribut `salle` ne peut-il pas être une clé primaire pour la relation `Ordinateur` ?

(b) On donne ci-dessous un extrait de la relation `Imprimante` :

Les 5 premières lignes de la relation `Imprimante`

nom_imprimante	marque_imp	modele_imp	salle	nom_ordi
imp_BTS_NB	HP	Laserjet pro M15w	114	Tech-62
imp_BTS_Couleur	Canon	Megatank Pixma G5050	114	Tech-62
imp_salle-info1	Brother	2360DN	223	Gen-132
imp_salle-info1	Brother	2360DN	223	Gen-133
imp_salle-info1	Brother	2360DN	223	Gen-134

On prend `(nom_imprimante, nom_ordi)` comme clé primaire. Écrire le schéma relationnel de la relation `Imprimante` en précisant les éventuelles clés étrangères pour les autres relations.

4. On donne ci-dessous un extrait de la relation `Videoprojecteur` :

Les 4 premières lignes de la relation `Videoprojecteur`

salle	marque_video	modele_video	tni
012	Epson	xb27	true
114	Sanyo	PLV-Z3	false
223	Optoma	HD143X	false
225	Optoma	HD143X	true

(a) Écrire une requête SQL pour ajouter à la relation `Videoprojecteur` le vidéoprojecteur nouvellement installé en salle 315 de marque NEC, modèle ME402X et non relié à un TNI.

(b) Écrire une requête SQL permettant de récupérer les attributs `salle, nom_ordi, marque_video` des ordinateurs connectés à un vidéoprojecteur équipé d'un TNI.

EXERCICE 2 (4 points)

Cet exercice porte sur les arbres binaires de recherche et la programmation orientée objet.

On rappelle qu'un arbre binaire est composé de nœuds, chacun des nœuds possédant éventuellement un sous-arbre gauche et éventuellement un sous-arbre droit. Un nœud sans sous-arbre est appelé feuille. La taille d'un arbre est le nombre de nœuds qu'il contient ; sa hauteur est le nombre de nœuds du plus long chemin qui joint le nœud racine à l'une des feuilles. Ainsi la hauteur d'un arbre réduit à un nœud, c'est-à-dire la racine, est 1.

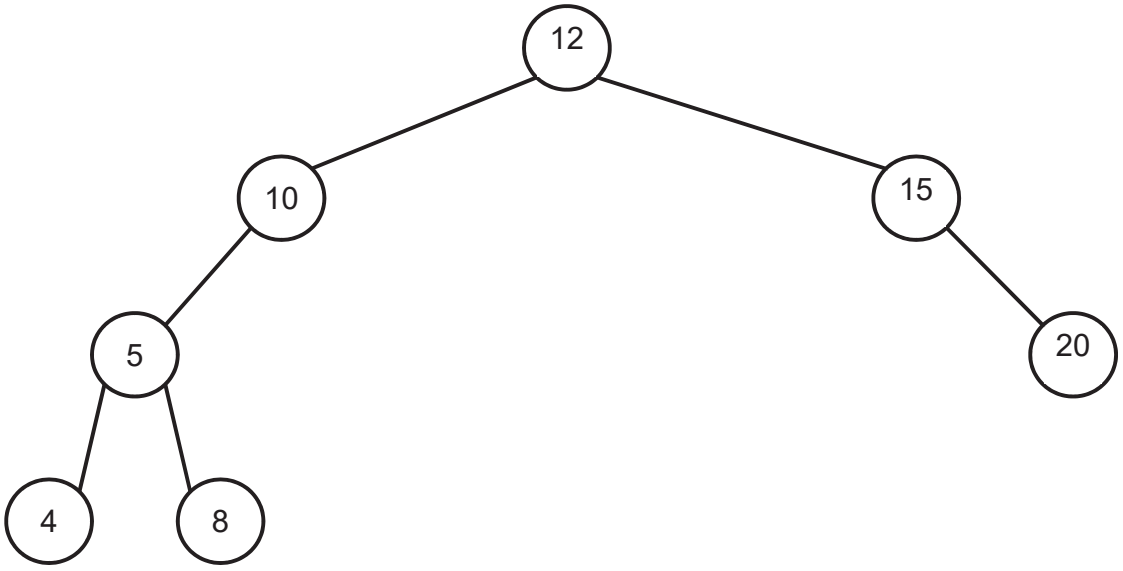
Dans un arbre binaire de recherche, chaque nœud contient une clé, ici un nombre entier, qui est :

- strictement supérieure à toutes les clés des nœuds du sous-arbre gauche ;
- strictement inférieure à toutes les clés des nœuds du sous-arbre droit.

Ainsi les clés de cet arbre sont toutes distinctes.

Un arbre binaire de recherche est dit « bien construit » s'il n'existe pas d'arbre de hauteur inférieure qui pourrait contenir tous ses nœuds.

On considère l'arbre binaire de recherche ci-dessous.



1. **a**. Quelle est la taille de l'arbre ci-dessus ?
   b. Quelle est la hauteur de l'arbre ci-dessus ?

2. Cet arbre binaire de recherche n'est pas « bien construit ». Proposer un arbre binaire de recherche contenant les mêmes clés et dont la hauteur est plus petite que celle de l'arbre initial.

3. Les classes `Noeud` et `Arbre` ci-dessous permettent de mettre en œuvre en Python la structure d'arbre binaire de recherche. La méthode `insere` permet d'insérer récursivement une nouvelle clé.

```
class Noeud :

    def __init__(self, cle):
        self.cle = cle
        self.gauche = None
        self.droit = None

    def insere(self, cle):
        if cle < self.cle :
            if self.gauche == None :
                self.gauche = Noeud(cle)
            else :
                self.gauche.insere(cle)
        elif cle > self.cle :
            if self.droit == None :
                self.droit = Noeud(cle)
            else :
                self.droit.insere(cle)

class Arbre :

    def __init__(self, cle):
        self.racine = Noeud(cle)

    def insere(self, cle):
        self.racine.insere(cle)
```

Donner la représentation de l'arbre codé par les instructions ci-dessous.

```
a = Arbre(10)
a.insere(20)
a.insere(15)
a.insere(12)
a.insere(8)
a.insere(4)
a.insere(5)
```

4. Pour calculer la hauteur d'un arbre non vide, on a écrit la méthode ci-dessous dans la classe `Noeud`.

```
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
```

Écrire la méthode `hauteur` de la classe `Arbre` qui renvoie la hauteur de l'arbre.

5. Écrire les méthodes `taille` des classes `Noeud` et `Arbre` permettant de calculer la taille d'un arbre.

6. On souhaite écrire une méthode `bien_construit` de la classe `Arbre` qui renvoie la valeur `True` si l'arbre est « bien construit » et `False` sinon.

On rappelle que la taille maximale d'un arbre binaire de recherche de hauteur h est $2^h - 1$.

a. Quelle est la taille minimale, notée t_min, d'un arbre binaire de recherche « bien construit » de hauteur h ?

b. Écrire la méthode `bien_construit` demandée.

EXERCICE 3 (4 points)

Cet exercice porte sur les tableaux et sur la programmation de base en Python.

On rappelle que `len` est une fonction qui prend un tableau en paramètre et renvoie sa longueur. C'est-à-dire le nombre d'éléments présents dans le tableau.
Exemple : `len([12, 54, 34, 57])` vaut 4.

Le but de cet exercice est de programmer différentes réductions pour un site de vente de vêtements en ligne.
On rappelle que si le prix d'un article avant réduction est de x euros,

- son prix vaut $0,5x$ si on lui applique une réduction de 50%,
- son prix vaut $0,6x$ si on lui applique une réduction de 40%,
- son prix vaut $0,7x$ si on lui applique une réduction de 30%,
- son prix vaut $0,8x$ si on lui applique une réduction de 20%,
- son prix vaut $0,9x$ si on lui applique une réduction de 10%.

Dans le système informatique du site de vente, l'ensemble des articles qu'un client veut acheter, appelé *panier*, est modélisé par un tableau de flottants.
Par exemple, si un client veut acheter un pantalon à 30,50 euros, un tee-shirt à 15 euros, une paire de chaussettes à 6 euros, une jupe à 20 euros, une paire de collants à 5 euros, une robe à 35 euros et un short à 10,50 euros, le système informatique aura le tableau suivant :

`tab = [30.5, 15.0, 6.0, 20.0, 5.0, 35.0, 10.5]`.

1. (a) Écrire une fonction Python `total_hors_reduction` ayant pour argument le tableau des prix des articles du panier d'un client et renvoyant le total des prix de ces articles.

   (b) Le site de vente propose la promotion suivante comme offre de bienvenue : 20% de réduction sur le premier article de la liste, 30% de réduction sur le deuxième article de la liste (s'il y a au moins deux articles) et aucune réduction sur le reste des articles (s'il y en a).
   Recopier sur la copie et compléter la fonction Python `offre_bienvenue` prenant en paramètre le tableau `tab` des prix des articles du panier d'un client et renvoyant le total à payer lorsqu'on leur applique l'offre de bienvenue.

```
def offre_bienvenue(tab):
    """ tableau -> float """
    somme=0
    longueur=len(tab)
    if longueur > 0 :
        somme=tab[0]*...
    if longueur > 1 :
        somme=somme + ...
    if longueur > 2 :
        for i in range(2,longueur):
            somme=...
    return ...
```

Pour toute la suite de l'exercice, on pourra utiliser la fonction `total_hors_reduction` même si la question 1 n'a pas été traitée.

2. Lors de la période des soldes, le site de vente propose les réductions suivantes :
   - si le panier contient 5 articles ou plus, une réduction globale de 50%,
   - si le panier contient 4 articles, une réduction globale de 40%,
   - si le panier contient 3 articles, une réduction globale de 30%,
   - si le panier contient 2 articles, une réduction globale de 20%,
   - si le panier contient 1 article, une réduction globale de 10%.

   Proposer une fonction Python `prix_solde` ayant pour argument le tableau `tab` des prix des articles du panier d'un client et renvoyant le total des prix de ces articles lorsqu'on leur applique la réduction des soldes.

3. (a) Écrire une fonction `minimum` qui prend en paramètre un tableau `tab` de nombres et renvoie la valeur minimum présente dans le tableau.

   (b) Pour ses bons clients, le site de vente propose une offre promotionnelle, à partir de 2 articles achetés, l'article le moins cher des articles commandés est offert.

   Écrire une fonction Python `offre_bon_client` ayant pour paramètre le tableau des prix des articles du panier d'un client et renvoyant le total à payer lorsqu'on leur applique l'offre bon client.

4. Afin de diminuer le stock de ses articles dans ses entrepôts, l'entreprise imagine faire l'offre suivante à ses clients : en suivant l'ordre des articles dans le panier du client, elle considère les 3 premiers articles et offre le moins cher, puis les 3 suivants et offre le moins cher et ainsi de suite jusqu'à ce qu'il reste au plus 2 articles qui n'ont alors droit à aucune réduction.

   Exemple : Si le panier du client contient un pantalon à 30,50 euros, un tee-shirt à 15 euros, une paire de chaussettes à 6 euros, une jupe à 20 euros, une paire de collants à 5 euros, une robe à 35 euros et un short à 10,50 euros, ce panier est représenté par le tableau suivant :

   ```
   tab = [30.5, 15.0, 6.0, 20.0, 5.0, 35.0, 10.5]
   ```

   Pour le premier groupe (le pantalon à 30,50 euros, le tee-shirt à 15 euros, la paire de chaussettes à 6 euros), l'article le moins cher, la paire de chaussettes à 6 euros, est offert. Pour le second groupe (la jupe à 20 euros, la paire de collants à 5 euros, la robe à 35 euros), la paire de collants à 5 euros est offerte.

   Donc le total après promotion de déstockage est 111 euros.

   On constate que le prix après promotion de déstockage dépend de l'ordre dans lequel se présentent les articles dans le panier.

   (a) Proposer un panier contenant les mêmes articles que ceux de l'exemple mais ayant un prix après promotion de déstockage différent de 111 euros.

   (b) Proposer un panier contenant les mêmes articles mais ayant le prix après promotion de déstockage le plus bas possible.

   (c) Une fois ses articles choisis, quel algorithme le client peut-il utiliser pour modifier son panier afin de s'assurer qu'il obtiendra le prix après promotion de déstockage le plus bas possible ? On ne demande pas d'écrire cet algorithme.

EXERCICE 4 (4 points)

Cet exercice traite de manipulation de tableaux, de récursivité et du paradigme « diviser pour régner ».

Dans un tableau Python d'entiers `tab`, on dit que le couple d'indices (i, j) forme une inversion lorsque $i < j$ et `tab[i]` $>$ `tab[j]`. On donne ci-dessous quelques exemples.

- Dans le tableau `[1, 5, 3, 7]`, le couple d'indices $(1,2)$ forme une inversion car $5 > 3$. Par contre, le couple $(1,3)$ ne forme pas d'inversion car $5 < 7$.
  Il n'y a qu'une inversion dans ce tableau.

- Il y a trois inversions dans le tableau `[1, 6, 2, 7, 3]`, à savoir les couples d'indices $(1, 2)$, $(1, 4)$ et $(3, 4)$.

- On peut compter six inversions dans le tableau `[7, 6, 5, 3]` : les couples d'indices $(0, 1)$, $(0, 2)$, $(0, 3)$, $(1, 2)$, $(1, 3)$ et $(2, 3)$.

On se propose dans cet exercice de déterminer le nombre d'inversions dans un tableau quelconque.

Questions préliminaires

1. Expliquer pourquoi le couple $(1, 3)$ est une inversion dans le tableau `[4, 8, 3, 7]`.
2. Justifier que le couple $(2, 3)$ n'en est pas une.

Partie A : Méthode itérative

Le but de cette partie est d'écrire une fonction itérative `nombre_inversion` qui renvoie le nombre d'inversions dans un tableau. Pour cela, on commence par écrire une fonction `fonction1` qui sera ensuite utilisée pour écrire la fonction `nombre_inversion`.

1. On donne la fonction suivante.

```
def fonction1(tab, i):
    nb_elem = len(tab)
    cpt = 0
    for j in range(i+1, nb_elem):
        if tab[j] < tab[i]:
            cpt += 1
    return cpt
```

   a. Indiquer ce que renvoie la `fonction1(tab, i)` dans les cas suivants.
      - Cas n°1 : `tab = [1, 5, 3, 7]` et `i = 0`.
      - Cas n°2 : `tab = [1, 5, 3, 7]` et `i = 1`.
      - Cas n°3 : `tab = [1, 5, 2, 6, 4]` et `i = 1`.

   b. Expliquer ce que permet de déterminer cette fonction.

2. En utilisant la fonction précédente, écrire une fonction `nombre_inversion(tab)` qui prend en argument un tableau et renvoie le nombre d'inversions dans ce tableau.
   On donne ci-dessous les résultats attendus pour certains appels.

```
>>> nombre_inversions([1, 5, 7])
0
>>> nombre_inversions([1, 6, 2, 7, 3])
3
>>> nombre_inversions([7, 6, 5, 3])
6
```

3. Quelle est l'ordre de grandeur de la complexité en temps de l'algorithme obtenu ? Aucune justification n'est attendue.

Partie B : Méthode récursive

Le but de cette partie est de concevoir une version récursive de la fonction `nombre_inversion`.
On définit pour cela des fonctions auxiliaires.

1. Donner le nom d'un algorithme de tri ayant une complexité meilleure que quadratique.

Dans la suite de cet exercice, on suppose qu'on dispose d'une fonction `tri(tab)` qui prend en argument un tableau et renvoie un tableau contenant les mêmes éléments rangés dans l'ordre croissant.

2. Écrire une fonction `moitie_gauche(tab)` qui prend en argument un tableau `tab` et renvoie un nouveau tableau contenant la moitié gauche de `tab`. Si le nombre d'éléments de `tab` est impair, l'élément du centre se trouve dans cette partie gauche.
   On donne ci-dessous les résultats attendus pour certains appels.

```
>>> moitie_gauche([])
[]
>>> moitie_gauche([4, 8, 3])
[4, 8]
>>> moitie_gauche ([4, 8, 3, 7])
[4, 8]
```

Dans la suite, on suppose qu'on dispose de la fonction `moitie_droite(tab)` qui renvoie la moitié droite sans l'élément du milieu.

3. On suppose qu'une fonction `nb_inv_tab(tab1, tab2)` a été écrite. Cette fonction renvoie le nombre d'inversions du tableau obtenu en mettant bout à bout les tableaux `tab1` et `tab2`, à condition que `tab1` et `tab2` soient triés dans l'ordre croissant.
On donne ci-dessous deux exemples d'appel de cette fonction :

```
>>> nb_inv_tab([3, 7, 9], [2, 10])
3
>>> nb_inv_tab([7, 9, 13], [7, 10, 14])
3
```

En utilisant la fonction `nb_inv_tab` et les questions précédentes, écrire une fonction récursive `nb_inversions_rec(tab)` qui permet de calculer le nombre d'inversions dans un tableau. Cette fonction renverra le même nombre que `nombre_inversions(tab)` de la partie A. On procédera de la façon suivante :

- Séparer le tableau en deux tableaux de tailles égales (à une unité près).
- Appeler récursivement la fonction `nb_inversions_rec` pour compter le nombre d'inversions dans chacun des deux tableaux.
- Trier les deux tableaux (on rappelle qu'une fonction de tri est déjà définie).
- Ajouter au nombre d'inversions précédemment comptées le nombre renvoyé par la fonction `nb_inv_tab` avec pour arguments les deux tableaux triés.

EXERCICE 5 (4 points)

Cet exercice porte sur la notion de pile, de file et sur la programmation de base en Python.

Les interfaces des structures de données abstraites `Pile` et `File` sont proposées ci-dessous.
On utilisera uniquement les fonctions ci-dessous :

Structure de données abstraite : Pile
Utilise : Élément, Booléen

Opérations :
• `creer_pile_vide : ∅ ⟶ Pile`
`creer_pile_vide()` renvoie une pile vide
• `est_vide : Pile ⟶ Booléen`
`est_vide(pile)` renvoie True si `pile` est vide, False sinon
• `empiler : Pile, Élément ⟶ ∅`
`empiler(pile, element)` ajoute `element` à la pile `pile`
• `depiler : Pile ⟶ Élément`
`depiler(pile)` renvoie l'élément au sommet de la `pile` en le retirant de la `pile`

Structure de données abstraite : File
Utilise : Élément, Booléen

Opérations :
• `creer_file_vide : ∅ ⟶ File`
`creer_file_vide()` renvoie une file vide
• `est_vide : File ⟶ Booléen`
`est_vide(file)` renvoie True si `file` est vide, False sinon
• `enfiler : File, Élément ⟶ ∅`
`enfiler(file, element)` ajoute `element` dans la file `file`
• `defiler : File ⟶ Élément`
`defiler(file)` renvoie l'élément au sommet de la file `file` en le retirant de la file `file`

1. (a) On considère la file `F` suivante :

enfilement ⟶ | "rouge" "vert" "jaune" "rouge" "jaune" | ⟶ défilement

Quel sera le contenu de la pile `P` et de la file `F` après l'exécution du programme Python suivant ?

```
1 P = creer_pile_vide()
2 while not(est_vide(F)):
3     empiler(P, defiler(F))
```

(b) Créer une fonction *taille_file* qui prend en paramètre une file `F` et qui renvoie le nombre d'éléments qu'elle contient. Après appel de cette fonction la file `F` doit avoir retrouvé son état d'origine.

```
def taille_file(F):
    """File -> Int"""
```

2. Écrire une fonction `former_pile` qui prend en paramètre une file `F` et qui renvoie une pile `P` contenant les mêmes éléments que la file.

   Le premier élément sorti de la file devra se trouver au sommet de la pile ; le deuxième élément sorti de la file devra se trouver juste en-dessous du sommet, etc.

   Exemple : si `F =` | "rouge" | "vert" | "jaune" | "rouge" | "jaune" | alors l'appel `former_pile(F)` va renvoyer la pile `P` ci-dessous :

   `P` =
"jaune"
"rouge"
"jaune"
"vert"
"rouge"

3. Écrire une fonction *nb_elements* qui prend en paramètres une file `F` et un élément `elt` et qui renvoie le nombre de fois où `elt` est présent dans la file `F`.

   Après appel de cette fonction la file `F` doit avoir retrouvé son état d'origine.

4. Écrire une fonction `verifier_contenu` qui prend en paramètres une file `F` et trois entiers : `nb_rouge`, `nb_vert` et `nb_jaune`.

   Cette fonction renvoie le booléen `True` si "rouge" apparaît au plus `nb_rouge` fois dans la file `F`, "vert" apparaît au plus `nb_vert` fois dans la file `F` et "jaune" apparaît au plus `nb_jaune` fois dans la file `F`. Elle renvoie `False` sinon. On pourra utiliser les fonctions précédentes.