CORRECTION BAC NSI 2025 AN J1

Exercice 1

1. D'après l'arbre de décision, on identifie un seul végétal : le sorbier.

2. Dans cet arbre, on ne peut identifier le végétal : la feuille a une étiquette vide.

3.
```
arbre_2 = Noeud("Simples ?",
                Feuille_resultat([]),
                Noeud("Alternées ?",
                       Noeud("Bord denté ?",
                             Feuille_resultat(["Sorbier"]),
                             Feuille_resultat(["Robinier", "Noyer"])
                            ),
                       Feuille_resultat([])
                      )
                )
```

4.
```
def est_resultat(self):
    return False
```

5.
```
def est_resultat(self):
    return True
```

6.
```
def nb_vegetaux(self):
    return len(self.vegetaux)
```

7.
```
def nb_vegetaux(self):
    return self.sioui.nb_vegetaux() + self.sinon.nb_vegetaux()
```

8.
```
def liste_questions(self):
    return []
```

9.
```
def liste_questions(self):
    return [self.question] + self.sioui.liste_questions()  \
                           + self.sinon.liste_questions()
```

10.
```
def est_bien_renseigne(dico_vegetal, arbre):
    for question in arbre.liste_questions() :
        if question not in dico_vegetal :
            return False
    return True
```

11.
```
def identifier_vegetaux(dico_vegetal, arbre):
    if arbre.est_resultat() :
        return arbre.vegetaux
    else:
        if dico_vegetal[arbre.question] :
            return identifier_vegetaux(dico_vegetal, arbre.sioui)
        else :
            return identifier_vegetaux(dico_vegetal, arbre.sinon)
```

Exercice 2

1. ```
   def passer_transit(self):
       self.etat = 'transit'
   ```

2. ```
   def ajouter_colis(liste, colis):
       if colis.poids <= 25:
           liste.append(colis)
       else :
           print("Dépassement du poids maximal autorisé")
   ```

3. ```
   def nb_colis(liste):
       return len(liste)
   ```

4. ```
   def poids_total(liste):
       total = 0
       for c in liste :
           total = total + c.poids
       return total
   ```

5. ```
   def liste_colis_etat(liste, statut):
       assert statut in ('préparé', 'transit', 'livré')
       return [colis for colis in liste if colis.etat == statut]
   ```

6. C'est un tri par sélection. Son coût est quadratique dans le pire des cas.

7. Le tri fusion a un coût en n×log(n).

8. ```
   def chargement_glouton(liste, rang, capacite):
       if rang == len(liste) :
           return []
       elif liste[rang].poids <= capacite:
           return [liste[rang]] + chargement_glouton(liste, rang+1,
                                       capacite - liste[rang].poids)
       else :
           return chargement_glouton(liste, rang+1, capacite)
   ```

9. On peut obtenir cette erreur car le nombre d'appels récursifs est limité par la taille de la pile d'appel. En Python c'est de l'ordre de 1000. Si la liste est trop longue on aura cette erreur.

10. ```
    def chargement_glouton2(liste, rang, capacite):
        """ algo iteratif """
        colis_a_charger = []
        i = 0
        while capacite > 0 and i < len(liste) :
            colis = liste[i]
            if colis.poids < capacite :
                colis_a_charger.append(colis)
                capacite = capacite - colis.poids
            i = i + 1
        return colis_a_charger
    ```
    ```

Exercice 3

Partie A

1. L'attribut annee est de type INTEGER.

2. Il serait pertinent d'avoir la contrainte *annee* compris dans l'intervalle :
[année courante – 18 ; annee courante]

3. L'attribut *num_parent* a une contrainte de référence car c'est une clé étrangère, qui se réfère à l'attribut *tel* de la table *parent*.

4. L'attribut *tel* de la table *parent* doit être la clé primaire, car c'est le seul qui identifie un parent de façon unique : noms et codes postaux peuvent être identiques pour 2 parents.

5. *Erreur du sujet : ils auraient dû écrire ce mauvais numéro dans la table enfant, sinon la requête ne lève pas d'erreur. Si ce mauvais numéro est dans la table enfant (et dans la table parent) alors on attendait la réponse :*
Cela lève une erreur car on ne peut pas modifier le *tel* auquel se réfère un attribut *num_parent* de la table enfant.

6.
```
UPDATE enfant
SET num_parent = 33619782812 WHERE num_parent = 33600682812 ;
DELETE FROM parent WHERE tel = 33619782812
```

7. Cette requête renvoie les prénoms des enfants née avant 2014, par ordre croissant d'année.

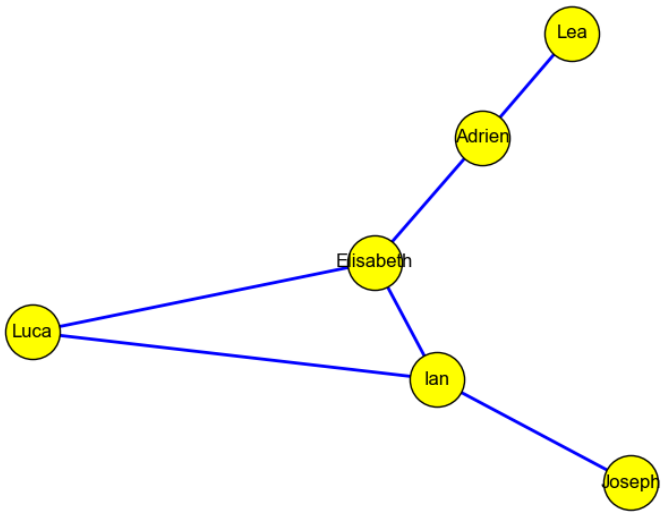
8.
```
SELECT prenom
FROM enfant
WHERE num_parent = 3619861122
ORDER BY prenom
```

9.
```
SELECT e.id, e.prenom
FROM enfant e JOIN parent p ON e.num_parent = p.tel
WHERE p.codep = 38520
```

Partie B

10. Le graphe est non-orienté car la relation de mésentente est forcément réciproque.

11.



12.
```
def degre(g, s) :
    return len(g[s])
```

13. (lignes 7 à 10):
```
    while j>=0 and degre(g, sommets[j]) < degre(g, sommet_courant):
        sommets[j+1] = sommets[j]
        j = j - 1
    sommets[j+1] = sommet_courant
```

14. Ce tri est un tri insertion, dans le pire cas il a un coût quadratique.

15.

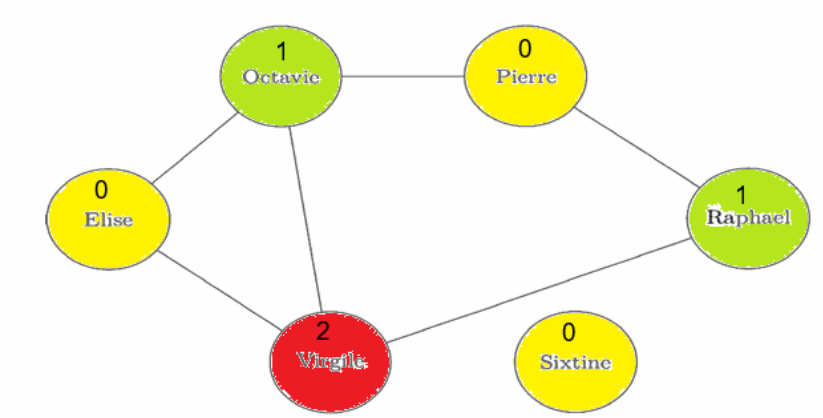


Figure 2. Graphe `g1`

16.
```
        couleur = plus_petite_couleur_hors_voisin(g, dc, s)
        dc[s] = couleur
```

17.
```
def welsh_powell(g):
    # initialisation à -1 pour tous les sommets dans dc
    dc = {s : -1 for s in g }
    # coloration
    for s in sommets_tries(g):
        couleur = plus_petite_couleur_hors_voisin(g, dc, s)
        dc[s] = couleur
    return dc
```

avec cet algorithme on obtient ceci pour g1 :
```
{'Elise': 2,
'Octavie': 0,
'Pierre': 1,
'Raphael': 0,
'Sixtine': 0,
'Virgile': 1}
```

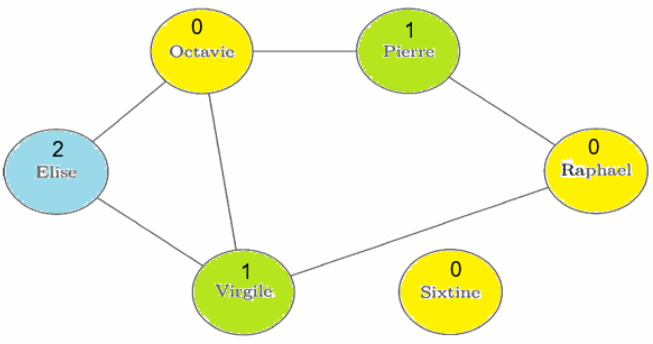


Figure 2. Graphe `g1`