2025 Amérique du nord jour 1

Exercice 1

1. Oui, Sorbier

2. Non, aucun végétal connu ne présente ces caractéristiques.

3.

```
feuille_S = Feuille_resultat(['Sorbier'])
feuille_RN = Feuille_resultat(['Robinier', 'Noyer'])
feuille_vide = Feuille_resultat([])

noeud_bd = Noeud('Bord denté ?', feuille_S, feuille_RN)
noeud_a = Noeud('Alternées ?', noeud_bd, feuille_vide)
arbre_2 = Noeud('Simples ?', feuille_vide, noeud_a)
```

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):
    # ajoute le colis à la fin de la liste s'il fait moins de 25kg
    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):
    liste_retour = []
    for c in liste:
        if c.etat == statut:
            liste_retour.append(c)
    return liste_retour
```

6. Tri par sélection. Son coût est en $O(n^2)$.

7. Tri par insertion avec un coût en $O(n^2)$.

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. S'il y a beaucoup de petits colis dans un grand camion il est possible qu'il y ait trop d'appels récursifs. La limite par défaut peut être à 1000.

10.

```
def chargement_glouton(liste, capacite):
    colis_a_charger = []
    for c in liste:
        if c.poids <= capacite:
            colis_a_charger.append(c)
            capacite = capacite - c.poids
    return colis_a_charger
```

Exercice 3

Partie A

1. INT

2. `annee` doit être positive.

3. `num_parent` est une clé étrangère. Elle doit donc faire référence à une valeur existante dans une autre table.

4. `tel` est un bon choix de clé primaire car deux parents ne peuvent pas avoir le même téléphone. Ce sera donc un champ unique. de plus, la clé étrangère `num_parent` faisant référence à `tel`, `tel` doit être unique.

5. Une erreur est levée car il doit y avoir un référence à l'ancienne valeur de téléphone dans le champ `num_parent` de la table téléphone. `num_parent` étant une clé étrangère faisant référence à la clé primaire `tel`.

6.

```sql
INSERT INTO parent VALUES ('Bauges', 33619782812, 73340);
UPDATE enfant SET num_parent = 33619782812 WHERE num_parent = 33600782812;
DELETE FROM parent WHERE tel = 33600782812;
```

7.

Nakamura
Hawa
Kian
Adrien

8.

```sql
SELECT prenom
FROM enfant
WHERE num_parent = 3619861122
ORDER BY prenom;
```

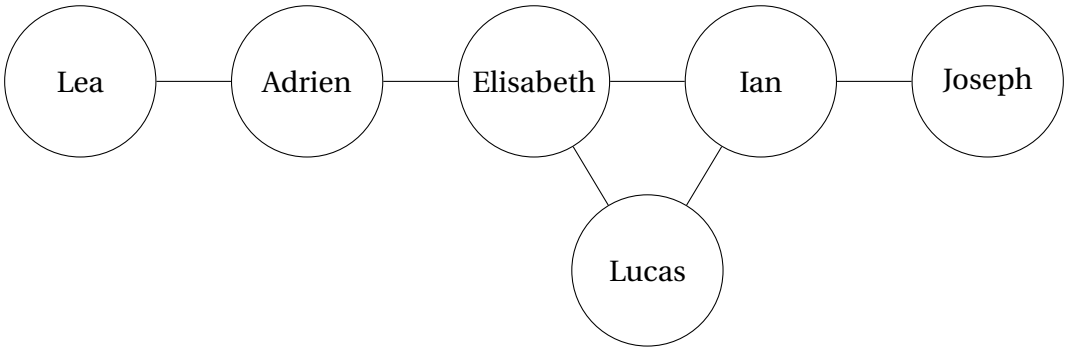
9.

```sql
SELECT id, prenom
FROM enfant
JOIN parent
ON num_parent = tel
WHERE codep = 38520;
```

Partie B

10. Le graphe peut être non-orienté car la mésentente est réciproque.

11.



12.

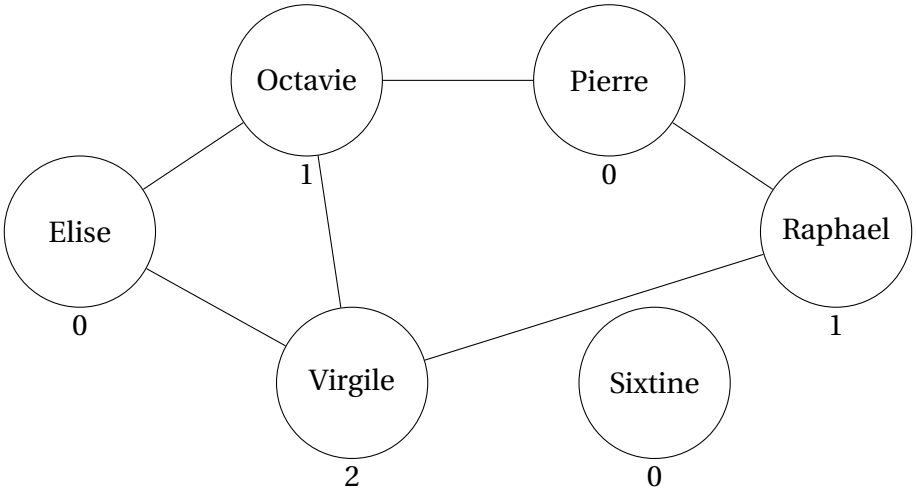
```python
def degre(g, s):
    return len(g[s])
```

13.

```python
def sommets_tries(g):
    sommets = [sommet for sommet in g]
    n = len(sommets)
    for i in range(1, n):
        sommet_courant = sommets[i]
        j = i-1
        while j>0 and degre(g, sommet_courant) > degre(g, sommets[j]):
            sommets[j+1] = sommets[j]
            j = j - 1
        sommets[j+1] = sommet_courant
    return sommets
```

14. C'est un tri par insertion. Son coût est quadratique.

15.



16.

```python
def colorer_graphe(g, dc):
    # Pré-condition : les clés de dc sont les sommets
    # de g, et les valeurs de dc sont toutes à -1
    for s in dc:
        couleur = plus_petite_couleur_hors_voisins(g, dc, s)
        dc[s] = couleur
```

17.

```python
def welsh_powell(g):
    # initialisation à -1 pour tous les sommets dans le dictionnaire dc
    dc = {}
    for s in g:
        dc[s] = -1
    # coloration en suivant l'approche de Welsh-Powell
    for s in sommets_tries(g):
        couleur = plus_petite_couleur_hors_voisins(g, dc, s)
        dc[s] = couleur
    return dc
```