Terminale NSI - DS 4 - Correction

Exercice 1

1. Nous avons un hôte d'adresse IP 172.16.2.3/24 qui appartient au réseau d'adresse réseau 172.16.2.0.
   C'est l'ordinateur d'Alice qui appartient à ce réseau.

2. cout = 10000/1000 = 10

3.

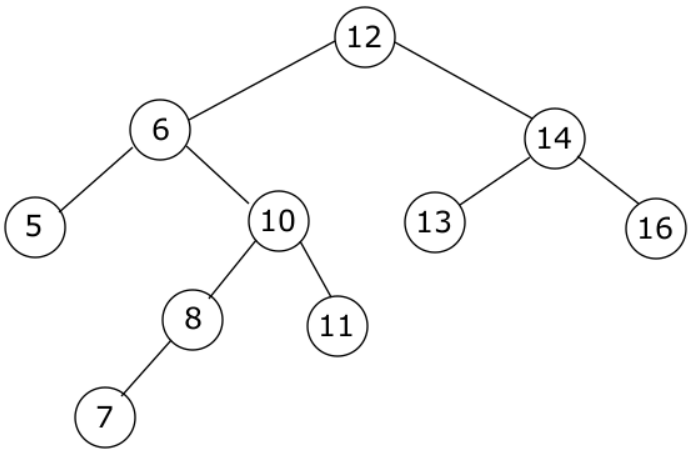
Routeur R6			
Destination	Pass	Cout	
LAN1	R5	21	car 10 + 11 = 21
LAN2	-	-	car directement connecté
WAN1	R5	11	car 10 + 1 = 11
WAN2	R5	20	car 10 + 10 = 20
WAN3	R5	11	car 10 + 1 = 11
WAN4	R5	12	car 10 + 2 = 12
WAN5	R5	10	car 10 pour aller en R5, puis directement connecté
WAN6	-	-	car directement connecté
WAN7	-	-	car directement connecté
WAN8	R5	10	car 10 pour aller en R5, puis directement connecté

4. Bob -> R1 -> R2 -> R5 -> R6 -> Alice

5. Le nouveau chemin est R1 -> R2 -> R4 - > R6. On évite le routeur R5, c'est donc le routeur 5 qui est en panne.

Exercice 2

1. Puisque les tâches doivent être extraites et exécutées dans le même ordre qu'elles ont été mémorisées, cela s'aligne avec le fonctionnement d'une file, respectant le principe FIFO.
2.
   a. il s'agit de la taille d'un arbre
   b. il s'agit de la racine de l'arbre
   c. il s'agit de la feuille d'un arbre
3.
   a. attributs de la classe Noeud : tache, indice, gauche et droite
   b. La méthode *insere* est dite récursive, car elle s'appelle elle-même.
      La méthode insere se termine, car à chaque appel récursif, on se déplace vers un sous-arbre d'un niveau inférieur. Puisque les arbres binaires ne peuvent pas avoir de boucles, chaque chemin de la racine vers une feuille est fini. Ainsi, la méthode finira par atteindre un sous-arbre vide où le nouveau nœud peut être inséré, ce qui arrête la récursivité.
   c. il s'agit du signe > (strictement supérieur)
   d.



4.

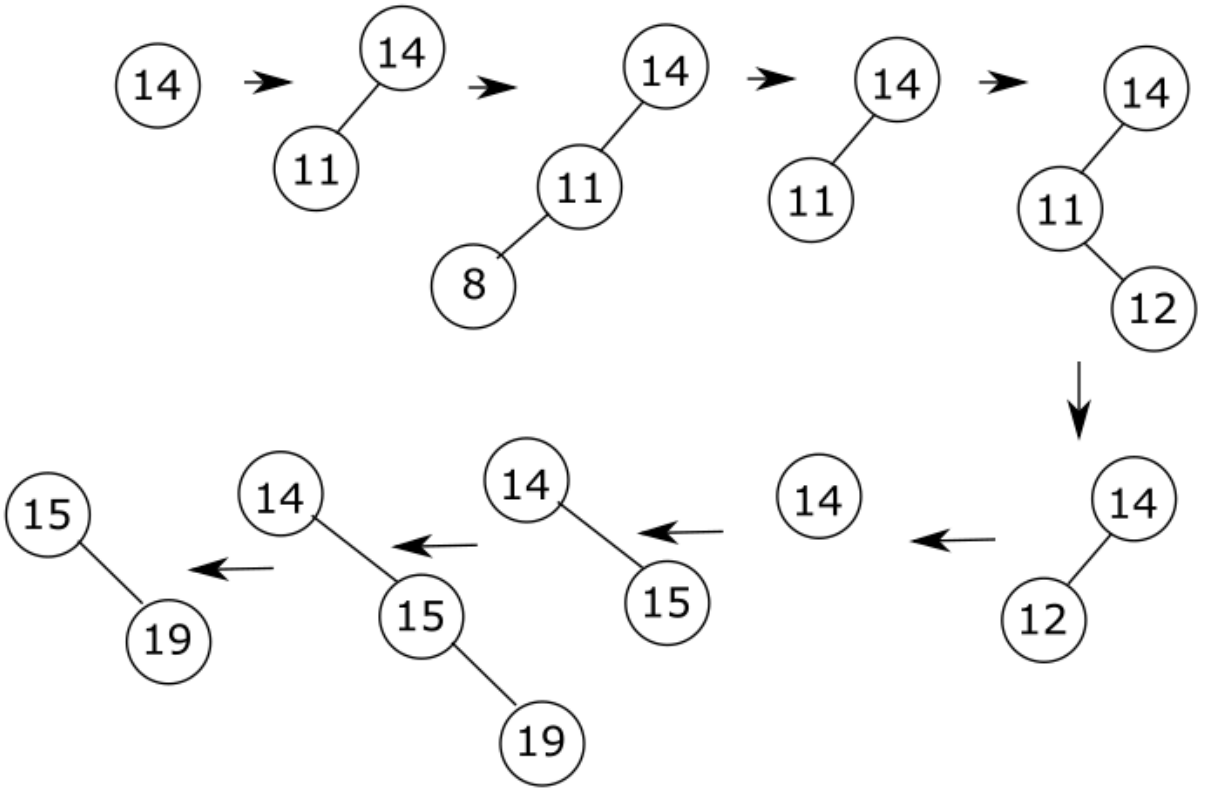
```
def est_present(self, indice_recherche) :
      """renvoie True si l'indice de priorité indice_recherche
      (int) passé en paramètre est déjà l'indice d'un nœud
      de l'arbre, False sinon"""
      if self.est_vide():
          return False
      if self.racine.indice == indice_recherche:
          return True
      if self.racine.indice > indice_recherche :
          return self.racine.gauche.est_present(indice_recherche)
      else :
          return self.racine.droite.est_present(indice_recherche)
```

5.
   a.
   parcours infixe : 6 - 8 - 10 - 12 - 13 - 14
   b.
   Le parcours infixe permet d'obtenir les valeurs des nœuds d'un arbre binaire de recherche dans un ordre croissant. Le parcours infixe va donc permettre d'obtenir les tâches à accomplir dans l'ordre des priorités

6.

```
    def tache_prioritaire(self):
            """renvoie la tache du noeud situé le plus
            à gauche de l'ABR supposé non vide"""
            if self.racine.gauche.est_vide(): #pas de nœud plus à
    gauche
                return self.racine.tache
            else:
                return self.racine.gauche.tache_prioritaire()
```

7.



1. Insertion de 14 : L'arbre commence avec un seul nœud, 14.
2. Insertion de 11 : Le nombre 11 est inséré comme enfant gauche de 14.
3. Insertion de 8 : Le nombre 8 est inséré comme enfant gauche de 11, formant une branche gauche.
4. Suppression de 8 : Le nœud le plus à gauche (8, qui est la tâche prioritaire) est supprimé.
   Le nœud11 devient maintenant la racine.
5. Insertion de 12 : Le nœud12 est inséré comme enfant droit de 11.
6. Suppression de 11 : Le nœud 11 est supprimé, et comme il a un sous-arbre droit, le 12 remplace le 11.
7. Suppression de 12 : Le nœud 12 est supprimé. L'arbre n'a plus qu'un seul nœud, le 14.
8. Insertion de 15 : Le nœud15 est inséré comme enfant droit de 14.
9. Insertion de 19 : Le nœud19 est inséré comme enfant droit de 15.
10. Suppression de 14 : Le nœud 14 est supprimé.
    Le nœud15, qui a un sous-arbre droit, remplace le 14 et devient la nouvelle racine.