Amérique du sud - septembre 2022 - sujet 2 (corrigé)

Exercice 1 (Programmation et algorithmique)

1. La fonction suivante convient :

```
def plus_proche_voisin(t, cible):
    dmin = distance(t[0], cible)
    idx_ppv = 0
    n = len(t)
    for idx in range(1, n):
        if distance(t[idx], cible) < dmin:
            dmin = distance(t[idx], cible)
            idx_ppv = idx
    return idx_ppv
```

2. Il n'y a pas de boucle imbriquée, ainsi la complexité en temps est linéaire selon n, soit en $O(n)$.

3. (a) Afin de ne faire qu'un seul appel à la fonction `distance(obj, cible)`, il suffit de sauvegarder la valeur retournée dans une variable au début de la boucle `for` :

```
for idx in range(n):
    d = distance(obj, cible)
```

et de remplacer `distance(obj, cible)` par `d` dans le reste du programme.

(a) Maintenir la liste `kppv` triée permet au moment de la sélection des k plus proches voisins (`kppv`) de prendre directement les k premiers de la liste. De plus, il est moins coûteux en temps de maintenir la liste triée que de tout trier à la fin du processus.

(b) La fonction suivante convient :

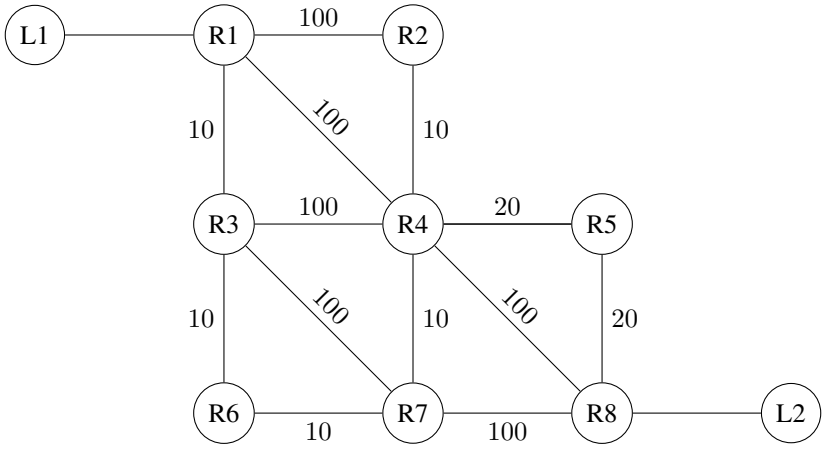
```
def insertion(kppv, idx, d):
    n = len(kppv)
    i = 0
    while i < n and kppv[i][1] < d:
        i += 1
    kppv.insert(i, (idx, d))
```

Exercice 2 (Réseaux et protocoles de routage)
Partie A.

1. La commande linux permettant d'avoir des informations sur l'interface réseau est `ipconfig`.
2. Le protocole qui permet d'attribuer automatiquement une adresse IP est `DHCP`.
3. Seules les adresses `192.168.1.1` et `192.168.1.255` font parties du réseau `192.168.1.14/24`. Néanmoins, celle se terminant par 255 est l'adresse du broadcast du réseau. Ainsi, la seule adresse valide pour un autre appareil est `192.168.1.1`.
4. De nos jours, les ordinateurs derrière une box ne possèdent pas une adresse internet propre et seule la box en possède une. Ainsi, c'est possible mais l'adresse `88.168.10.210` serait vraissemblablement celle de sa box.
5. Oui, car les adresses `192.168.x.x` sont des adresses internes au réseau de l'élève et/ou du lycée, ainsi non routées sur internet.

Partie B.

1. Le coût d'une liaison VDSL est $\dfrac{10^9}{50 \times 10^6} = \dfrac{100}{5} = 20$
2. (a) On a le schéma suivant :



(b) Du réseau L1 à L2, le protocole OSPF entraînera le chemin de poids minimal 80, passant par L1 – R1 – R3 – R6 – R7 – R4 – R5 – R8 – L2.

(c) Pour passer de R1 à R4 sans passer par R3-R6-R7, il faut que la liaison ne dépasse pas un coût de 40, soit un débit supérieur à $\frac{10^9}{40} = 25 \times 10^6$ b/s $= 25$ Mb/s.

Exercice 3 (Bases de données et SQL)

1. La requête suivante convient :

```
UPDATE ModeleVelo SET stock=0 WHERE nomModele='Bovelo'
```

2. Puisqu'il s'agit d'un nouveau fabricant, il faut effectuer la requête 4 puis la requête 2.
3. (a) La requête suivante convient :

```
SELECT nomModele, idFabricant FROM ModeleVelo WHERE stock = 0
```

(b) La requête suivante convient :

```
def parcours_maladies(arb):
    if arb == {}:
        return None
    parcours_maladies(arb['sag'])
    parcours_maladies(arb['sad'])
    if len(arb['sag'])==0 and len(arb['sad'])==0:
        print(arb['etiquette'])
```

(c) La requête suivante convient :

```
SELECT nom FROM Fabricant
JOIN ModeleVelo
ON ModeleVelo.idFabricant = Fabricant.idFabricant
WHERE stock > 0
```

4. La requête SQL permet d'obtenir le nom des clients ayant commandé un modèle de vélo `'Bovelo'` (si un même client a acheté plusieurs fois ce type de vélo, son nom n'apparaîtra qu'une seule fois).

Exercice 4 (Programmation et récursivité)

1. (a) On peut saisir la commande `from math import sqrt`

   (b) La fonction suivante convient :

```
def distance_points(a, b):
    return sqrt((b[0] - a[0])**2 + (b[1] - a[1])** 2)
```

2. La fonction suivante convient :

```
def distance(p, a, b):
    if a == b: # ou distance_points(a,b) == 0
        return distance_points(p, a)
    else:
        return distance_point_droite(p, a, b)
```

3. La fonction suivante convient :

```
def le_plus_loin(ligne):
    n = len(ligne)
    deb = ligne[0]
    fin = ligne[n-1]
    dmax = 0
    indice_max = 0
    for idx in range(1, n-1):
        p = ligne[idx]
        d = distance(p, deb, fin)
        if d > dmax:
            dmax = d
            indice_max = idx
    return (indice_max, dmax)
```

4. La fonction suivante convient :

```
def extrait(tab, i, j):
    ntab = []
    for k in range(i, j+1):
        ntab.append(tab[k])
    return ntab
```

5. La fonction suivante convient :

```
def simplifie(ligne, seuil):
    n = len(ligne)
    if n <= 2:
        return ligne
    else:
        indice_max, dmax = le_plus_loin(ligne)
        if dmax <= seuil:
            return [ligne[0], ligne[n-1]]
        else:
            seq1 = simplifie(extrait(ligne, 0, indice_max))
            seq2 = simplifie(extrait(ligne, indice_max, n-1))
            return seq1 + seq2
```

Exercice 5 (Arbres binaires, POO et récursivité)

1. La plus grande somme racine-feuille est $2 + 7 + 4 + 3 = 16$.
2. (a) La suite d'instructions suivante convient :

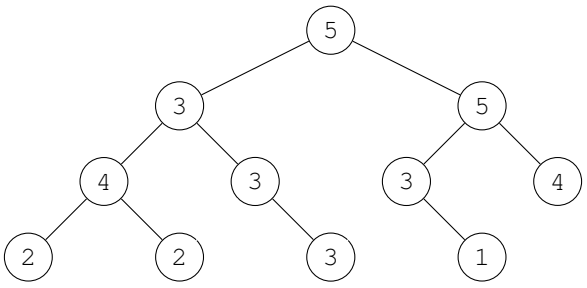
```
n1 = Noeud(1)
n4 = Noeud(4)
n7 = Noeud(7)
n7.modifier_sag(n4)
n7.modifier_sad(n1)
n8 = Noeud(8)
n5 = Noeud(5)
n5.modifier_sad(n8)
n2 = Noeud(2)
n2.modifier_sag(n7)
n2.modifier_sad(n5)
```

(a) La méthode `niveau` appelé sur `n2` renvoie 2.

3. La fonction suivante convient :

```
def pgde_somme(self):
    somg, somd = 0, 0
    if self.sag is not None :
        somg = self.sag.pgde_somme()
    if self.sad is not None :
        somd = self.sad.pgde_somme()
    return self.etiquette + max(somg, somd)
```

4. (a) On a l'arbre suivant :



(b) La fonction suivante convient :

```
def est_magique(self):
    if self.sag is not None and self.sad is not None :
        if not (self.sag.est_magique() and self.sad.est_magique()):
            return False # l'un des enfants n'est pas magique
        else: # les deux enfants sont magiques, renvoient-ils la même somme ?
            return self.sag.pgde_somme() == self.sad.pgde_somme()
    elif self.sag is not None: # magique si le seul enfant l'est
        return self.sag.est_magique()
    elif self.sad is not None: # magique si le seul enfant l'est
        return self.sad.est_magique()
    else: # une feuille est magique
        return True
```