

[← Index des sujets 2022](#)

## 22-NSIJ2AS1 : Corrigé

Année : **2022**

Centre : **Amérique du sud**

Jour : **2**

Enoncé : [!\[\]\(e3f8612927870f2e0f9f5989e6dd3064\_img.jpg\) PDF](#)

### 1. Exercice 1

*programmation, algorithmique et complexité*

- Le bloc d'instructions à écrire est surligné



Script Python

```

1 def plus_proche_voisin(t, cible):
2     dmin = distance(t[0],cible)
3     idx_ppv = 0
4     n = len(t)
5     for idx in range(1,n):
6         if distance(t[idx],cible) < dmin:
7             dmin = distance(t[idx],cible)
8             idx_ppv = idx
9     return idx_ppv

```



#### Note

On parcourt le tableau `t`, si un élément plus proche de la cible est trouvé alors on met à jour l'indice et la valeur du minimum.

- Le bloc est répété `n-1` fois où `n` est la taille du tableau `t`, comme le coût du bloc est constant, la complexité de la fonction `plus_proche_voisin` est linéaire (c'est à dire en  $\mathcal{O}(n)$ ).

3.  Note

Bien comprendre l'algorithme proposé :

- la liste `kppv` est initialement vide, on y rangera au fil du parcours les `k` plus proches voisins dans l'ordre *décroissant* de leur distance (l'élément d'indice 0 est donc le plus éloigné)
- si la liste `kppv` contient moins de `k` éléments, alors on y range l'élément parcouru
- sinon si un élément plus proche est trouvé alors on supprime le premier de la liste et on insère ce nouvel élément dans `kppv`.

a. Pour faire un seul appel à la fonction `distance`, il suffit de stocker le résultat de l'appel dans une variable :

```
d_idx = distance(obj,cible)
```

b. Maintenir la liste `kppv` triée permet de comparer uniquement avec son premier élément pour savoir si on insère.

c.

 Script Python

```
1 def insertion(kppv,idx,d):
2     position_insertion = 0
3     while d < kppv[position_insertion][1] and position_insertion<len(kppv)-1:
4         position_insertion += 1
5     kppv.insert(position_insertion,(idx,d))
```

## 2. Exercice 2

réseaux et routage

### Partie A

1. C'est la commande `ifconfig`

 Note

Pour mémoire :

- `ping` permet de tester l'accès à une machine à travers un réseau IP
- `ps` liste les processus
- `ls` liste les fichiers et dossiers

2. C'est le protocole DHCP

 Note

Pour mémoire :

- Un serveur DNS permet d'associer des noms de domaines à des adresses
- Le protocole TCP est le protocole de la couche transport du modèle TCP/IP chargé d'acheminer les informations (par paquets)
- Le protocole HTTP est le protocole de la couche application

3. La seule adresse IP possible est 192.168.1.1

 Note

- Le masque de sous réseau est 255.255.255.0, donc pour faire partie du même réseau, les trois premiers octets doivent être identiques. Les adresses 192.168.0.14 et 192.168.0.1 ne sont donc pas possibles (car elles ne commencent pas par 192.168.1)
- L'adresse 192.168.1.255 est une adresse réservée (adresse de diffusion ou broadcast en anglais)

4. C'est possible et cette adresse serait celle de la box vers Internet.

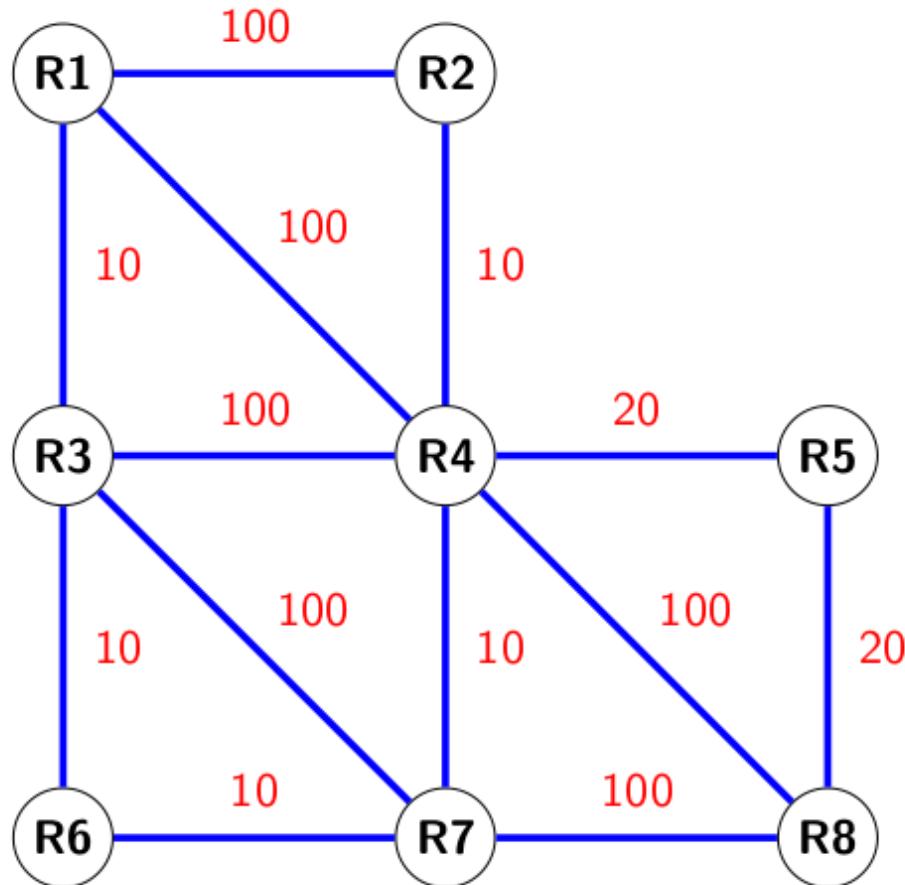
 Note

La box sert de routeur pour accéder à Internet

5. Oui, car les adresses 192.168.x.x ne sont pas routées sur internet

**Partie B**

1. La bande passante d'une liaison VDSL est 50 Mb/s, son coût est donc :  $\frac{10^9}{50 \times 10^6}$  ce qui fait bien 20.
- 2.



3. La route utilisée sera : R<sub>1</sub> → R<sub>3</sub> → R<sub>6</sub> → R<sub>7</sub> → R<sub>4</sub> → R<sub>5</sub> → R<sub>8</sub> pour un coût total de  $10 + 10 + 10 + 10 + 20 + 20 = 80$

4. Le coût maximal de cette liaison devra être de 40, en effet le coût maximal de la route R<sub>1</sub> → R<sub>4</sub> → R<sub>5</sub> → R<sub>8</sub> sera alors de 80. On doit résoudre  $\frac{10^9}{BP} < 40$  ce qui donne :  $BP > \frac{10^9}{40}$  c'est à dire  $BP > 25$  Mb/s.

### 3. Exercice 3

## *base de données*

- ## 1. Requête SQL

```
UPDATE ModeleVelo  
SET Stock = 0  
WHERE nomModele = 'Bovelo';
```

## Attention

La clé primaire de la table ModeleVelo est `idModele`, en toute logique les opérations de mises à jour devraient s'effectuer via cette clé. Par exemple si plusieurs vélos différents ont comme nomModele `Bovelo` les stocks de ces vélos seront tous mis à zéro.

2. On doit commencer par exécuter la requête 4 ( Ravel est un nouveau fabricant) puis la requête 2.
3. a. Pour avoir les modèles en rupture de stock et l'identifiant de leur fabricant :

 Requête SQL

```
SELECT nomModele, idFabricant
FROM ModeleVelo
WHERE Stock = 0;
```

- b. Pour avoir le nombre de commandes passées depuis le 2022-01-01 inclus :

 Requête SQL

```
SELECT COUNT(*)
FROM Commande
WHERE date >= '2022-01-01';
```

- c. Pour avoir les noms des fabricants dont le stock de vélos est strictement positif.

 Requête SQL

```
SELECT DISTINCT Fabricant.nom FROM Fabricant
JOIN ModeleVelo ON ModeleVelo.idFabricant = Fabricant.idFabricant
WHERE ModeleVelo.Stock > 0;
```

4. Cette requête permet d'obtenir les noms des clients ayant commandé un vélo dont le modèle est Bovelo .

## 4. Exercice 4

*programmation en Python, récursivité et méthode diviser pour régner*

1. a. Pour importer la fonction `sqrt` du module `math`, on peut écrire :

 Script Python

```
from math import sqrt
```

- b.

 Script Python

```
1 def distance(a,b):
2     xa,ya = a
3     xb,yb = b
4     return sqrt((xb-xa)**2+(yb-ya)**2)
```

 Note

On rappelle que `a` est un tuple représentant les coordonnées du point `xa,ya = a` permet de récupérer ces deux coordonnées. Par exemple si `a = (5, 7)` alors `xa = 5` et `ya = 7`. On pourrait de façon équivalent écrire : `xa = a[0]` et `ya = a[1]`.

2.  Script Python

```

1 def distance(p,a,b):
2     if a == b:
3         return distance(p,a)
4     else:
5         return distance_point_droite(p, a, b)

```

3.  Script Python

```

1 def le_plus_loin(ligne):
2     n = len(ligne)
3     deb = ligne[0]
4     fin = ligne[n-1]
5     dmax = 0
6     indice_max = 0
7     for idx in range(1,n-1):
8         p = ligne[idx]
9         d = distance(p, deb, fin)
10    if d > dmax:
11        dmax = d
12        indice_max = idx
13    return indice_max, dmax

```

4.  Script Python

```

1 def extrait(tab, i, j):
2     return [tab[k] for k in range(i,j+1)]

```

5.  Script Python

```

1 def simplifie(ligne,seuil):
2     n = len(ligne)
3     if n <=2:
4         return ligne
5     else:
6         indice_max, dmax = le_plus_loin(ligne)
7         if dmax <= seuil:
8             return [ligne[0],ligne[n-1]]
9         else:
10            return simplifie(extrait(ligne,0,indice_max)) +
11            simplifie(extrait(ligne,indice_max,n))

```

## 5. Exercice 5

arbres binaires, programmation orientée objet et récursivité

- La plus grande somme racine-feuille de cet arbre est **16**, elle est obtenue pour la branche en rouge dans le schéma suivant :

```

graph TD
A["2"] --> B["7"]
A --> C["5"]
B --> D["4"]
B --> E["1"]
C --> F["3"]
C --> G["8"]
D --> H["2"]
D --> I["3"]
E --> V1[" "]
E --> J["5"]
F --> V2[" "]
F --> K["1"]
style V1 fill:#FFFFFF, stroke:#FFFFFF
style V2 fill:#FFFFFF, stroke:#FFFFFF
linkStyle 8 stroke:#FFFFFF,stroke-width:0px
linkStyle 10 stroke:#FFFFFF,stroke-width:0px
linkStyle 0 stroke:#FF0000,stroke-width:2px
linkStyle 2 stroke:#FF0000,stroke-width:2px
linkStyle 7 stroke:#FF0000,stroke-width:2px

```

- a. On peut écrire la suite d'instructions suivante :

### Script Python

```

s2 = Noeud(2)
s7 = Noeud(7)
s5 = Noeud(5)
s2.modifier_sag(s7)
s2.modifier_sad(s5)
s4 = Noeud(4)
s1 = Noeud(1)
s7.modifier_sag(s4)
s7.modifier_sad(s1)
s8 = Noeud(8)
s5.modifier_sad(s8)

```

- b. L'appel à niveau sur cet arbre renvoie 3.

### Script Python

```

def pgde_somme(self):
    if self.sag != None and self.sad!=None:
        pgde_gauche = self.sag.pgde_somme()
        pgde_droite = self.sad.pgde_somme()
        return self.etiquette + max(pgde_gauche, pgde_droite)
    if self.sag != None:
        return self.etiquette + self.sag.pgde_somme()
    if self.sad != None:
        return self.etiquette + self.sad.pgde_somme()
    return self.etiquette

```

## 4. a. Arbre complété :

```

graph TD
A["5"] --> B["3"]
A --> C["5"]
B --> D["4"]
B --> E["3"]
C --> F["3"]
C --> G["4"]
D --> H["2"]
D --> I["2"]
E --> V1[" "]
E --> J["3"]
F --> V2[" "]
F --> K["1"]
style V1 fill:#FFFFFF, stroke:#FFFFFF
style V2 fill:#FFFFFF, stroke:#FFFFFF
style B fill:#CCCCCC, stroke:#0000FF
style E fill:#CCCCCC, stroke:#0000FF
style G fill:#CCCCCC, stroke:#0000FF
style I fill:#CCCCCC, stroke:#0000FF
linkStyle 8 stroke:#FFFFFF,stroke-width:0px
linkStyle 10 stroke:#FFFFFF,stroke-width:0px

```

b.

 Script Python

```

def est_magique(self):
    if self.sad is not None and self.sag is not None:
        return self.sad.est_magique() and self.sag.est_magique() and self.sag.pgde_somme() ==
self.sad.pgde_somme()
    elif self.sad is not None:
        return self.sad.est_magique()
    elif self.sag is not None:
        return self.sag.est_magique()
    else:
        return True

```