

Terminale NSI - DS 4

EXERCICE 1 (7 points)

L'exercice porte sur l'architecture matérielle, les réseaux et les systèmes d'exploitation.

Nous allons étudier les communications entre Bob et Alice. Ils communiquent au travers du réseau ci-dessous dont le protocole de routage est le protocole OSPF qui minimise le cout des communications :

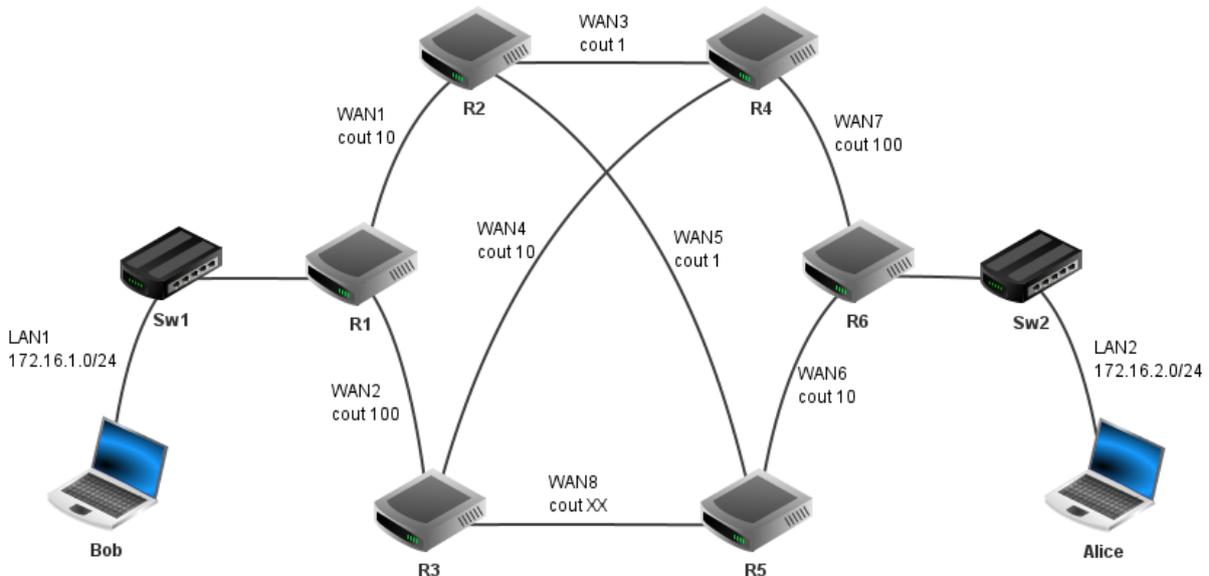


Figure 1 : Plan du réseau de communication entre Alice et Bob.

LAN : réseau local ; WAN : réseau étendu ; R : routeur ; Sw : Switch

Une adresse IPv4 est composée de quatre octets soit 32 bits. Une adresse de sous-réseau avec la notation /n signifie que les n premiers bits de l'adresse correspondent à la partie « réseau » et les suivants à la partie « machine ».

L'adresse dont tous les bits de la partie « machine » sont à 0 est appelée adresse du réseau.

L'adresse dont tous les bits de la partie « machine » sont à 1 est appelée adresse de diffusion.

Ces adresses sont réservées et ne peuvent pas être attribuées à des machines.

Le choix des routes empruntées par les paquets IP est uniquement basé sur le protocole OSPF. On prendra comme débit maximal de référence 10 000 Mbit/s.

Le cout est alors calculé de la façon suivante :

$$\text{cout} = \frac{\text{débit maximal de référence}}{\text{débit du réseau concerné}}$$

1. La configuration IP partielle ci-dessous a été affichée sur l'un des ordinateurs :

```
IP hôte : 172.16.2.3
IP passerelle : 172.16.2.253
```

Indiquer en justifiant si cette configuration appartient à l'ordinateur de Bob ou d'Alice.

2. Le réseau WAN8 a un débit de 1 000 Mbit/s. Calculer le cout correspondant.

3. On donne les tables de routage des routeurs R1 à R5, dans lesquelles Pass. désigne la passerelle (qui correspond au routeur suivant) :

Routeur R1			Routeur R2			Routeur R3		
Destination	Pass.	Cout	Destination	Pass.	Cout	Destination	Pass.	Cout
LAN1	-	-	LAN1	R1	10	LAN1	R4	21
LAN2	R2	21	LAN2	R5	11	LAN2	R5	20
WAN1	-	-	WAN1	-	-	WAN1	R4	11
WAN2	-	-	WAN2	R1	10	WAN2	-	-
WAN3	R2	10	WAN3	-	-	WAN3	R4	10
WAN4	R2	11	WAN4	R4	1	WAN4	-	-
WAN5	R2	10	WAN5	-	-	WAN5	R5	10
WAN6	R2	11	WAN6	R5	1	WAN6	R5	10
WAN7	R2	11	WAN7	R4	1	WAN7	R4	10
WAN8	R2	11	WAN8	R5	1	WAN8	-	-

Routeur R4			Routeur R5		
Destination	Pass.	Cout	Destination	Pass.	Cout
LAN1	R2	11	LAN1	R2	11
LAN2	R2	12	LAN2	R6	10
WAN1	R2	1	WAN1	R2	1
WAN2	R3	10	WAN2	R3	10
WAN3	-	-	WAN3	R2	1
WAN4	-	-	WAN4	R2	2
WAN5	R2	1	WAN5	-	-
WAN6	R2	2	WAN6	-	-
WAN7	-	-	WAN7	R2	2
WAN8	R2	2	WAN8	-	-

Figure 2 : Tables de routage des routeurs R1 à R5

Écrire sur votre copie la table de routage du routeur R6, en justifiant le Cout par un calcul pour chaque destination.

4. Bob envoie un message à Alice.

Énumérer dans l'ordre tous les routeurs par lesquels transitera ce message.

5. Un routeur tombe en panne, le nouveau cout pour la route entre Bob et Alice est de 111. Déterminer le nom du routeur en panne.

EXERCICE 2 (13 points)

L'exercice porte sur les arbres binaires de recherche et la programmation objet.

Dans un entrepôt de e-commerce, un robot mobile autonome exécute successivement les tâches qu'il reçoit tout au long de la journée.

La mémorisation et la gestion de ces tâches sont assurées par une structure de données.

1. Dans l'hypothèse où les tâches devraient être extraites de cette structure (pour être exécutées) dans le même ordre qu'elles ont été mémorisées, préciser si ce fonctionnement traduit le comportement d'une file ou d'une pile. Justifier.

En réalité, selon l'urgence des tâches à effectuer, on associe à chacune d'elles, lors de la mémorisation, un indice de priorité (nombre entier) distinct : il n'y a pas de valeur en double.

Plus cet indice est faible, plus la tâche doit être traitée prioritairement.

La structure de données retenue est assimilée à un arbre binaire de recherche (ABR) dans lequel chaque nœud correspond à une tâche caractérisée par son indice de priorité.

Rappel : Dans un arbre binaire de recherche, chaque nœud est caractérisé par une valeur (ici l'indice de priorité), telle que chaque nœud du sous-arbre gauche a une valeur strictement inférieure à celle du nœud considéré, et que chaque nœud du sous-arbre droit possède une valeur strictement supérieure à celle-ci. Cette structure de données présente l'avantage de mettre efficacement en œuvre l'insertion ou la suppression de nœuds, ainsi que la recherche d'une valeur.

Par exemple, le robot a reçu successivement, dans l'ordre, des tâches d'indice de priorité 12, 6, 10, 14, 8 et 13. En partant d'un arbre binaire de recherche vide, l'insertion des différentes priorités dans cet arbre donne la figure 1.

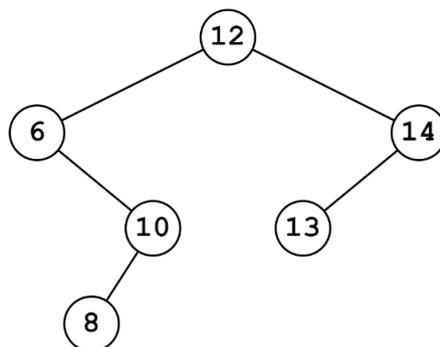


Figure 1 : Exemple d'un arbre binaire

2. En utilisant le vocabulaire couramment utilisé pour les arbres, préciser le terme qui correspond :

- a. au nombre de tâches restant à effectuer, c'est-à-dire le nombre total de nœuds de l'arbre ;
- b. au nœud représentant la tâche restant à effectuer la plus ancienne ;

c. au nœud représentant la dernière tâche mémorisée (la plus récente).

3. Lorsque le robot reçoit une nouvelle tâche, on déclare un nouvel objet, instance de la classe `Noeud`, puis on l'insère dans l'arbre binaire de recherche (instance de la classe `ABR`) du robot. Ces 2 classes sont définies comme suit :

```
1 class Noeud:
2     def __init__(self, tache, indice):
3         self.tache = tache #ce que doit accomplir le robot
4         self.indice = indice #indice de priorité (int)
5         self.gauche = ABR() #sous-arbre gauche vide (ABR)
6         self.droite = ABR() #sous-arbre droit vide (ABR)
7
8
9 class ABR:
10     #arbre binaire de recherche initialement vide
11     def __init__(self):
12         self.racine = None #arbre vide
13         #Remarque : si l'arbre n'est pas vide, racine est
14         #une instance de la classe Noeud
15
16     def est_vide(self):
17         """renvoie True si l'arbre autoréférencé est vide,
18         False sinon"""
19         return self.racine == None
20
21     def insere(self, nouveau_noeud):
22         """insere un nouveau noeud, instance de la classe
23         Noeud, dans l'ABR"""
24         if self.est_vide():
25             self.racine = nouveau_noeud
26         elif self.racine.indice ..... nouveau_noeud.indice
27             self.racine.gauche.insere(nouveau_noeud)
28         else:
29             self.racine.droite.insere(nouveau_noeud)
```

- Donner les noms des attributs de la classe `Noeud`.
- Expliquer en quoi la méthode `insere` est dite récursive et justifier rapidement qu'elle se termine.
- Indiquer le symbole de comparaison manquant dans le test à la **ligne 26** de la méthode `insere` pour que l'arbre binaire de recherche réponde bien à la définition de l'encadré « **Rappel** » de la page 6.
- On considère le robot dont la liste des tâches est représentée par l'arbre de la figure 1. Ce robot reçoit, successivement et dans l'ordre, des tâches d'indice de priorité 11, 5, 16 et 7, sans avoir accompli la moindre tâche entretemps. Recopier et compléter la figure 1 après l'insertion de ces nouvelles tâches.

4. Avant d'insérer une nouvelle tâche dans l'arbre binaire de recherche, il faut s'assurer que son indice de priorité n'est pas déjà présent.

Écrire une méthode `est_present` de la classe `ABR` qui répond à la description :

```
41 def est_present(self, indice_recherche) :
42     """renvoie True si l'indice de priorité indice_recherche
43     (int) passé en paramètre est déjà l'indice d'un nœud
44     de l'arbre, False sinon"""
```

5. Comme le robot doit toujours traiter la tâche dont l'indice de priorité est le plus petit, on envisage un parcours infixe de l'arbre binaire de recherche.

- a. Donner l'ordre des indices de priorité obtenus à l'aide d'un parcours infixe de l'arbre binaire de recherche de la **figure 1**.
- b. Expliquer comment exploiter ce parcours pour déterminer la tâche prioritaire.

6. Afin de ne pas parcourir tout l'arbre, il est plus efficace de rechercher la tâche du nœud situé le plus à gauche de l'arbre binaire de recherche : il correspond à la tâche prioritaire.

Recopier et compléter la méthode récursive `tache_prioritaire` de la classe `ABR`:

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

7. Une fois la tâche prioritaire effectuée, il est nécessaire de supprimer le nœud correspondant pour que le robot passe à la tâche suivante :

- Si le nœud correspondant à la tâche prioritaire est une feuille, alors il est simplement supprimé de l'arbre (cette feuille devient un arbre vide)
- Si le nœud correspondant à la tâche prioritaire a un sous-arbre droit non vide, alors ce sous-arbre droit remplace le nœud prioritaire qui est alors écrasé, même s'il s'agit de la racine.

Dessiner alors, pour chaque étape, l'arbre binaire de recherche (seuls les indices de priorités seront représentés) obtenu pour un robot, initialement sans tâche, et qui a, successivement dans l'ordre :

- étape 1 : reçu une tâche d'indice de priorité 14 à accomplir
- étape 2 : reçu une tâche d'indice de priorité 11 à accomplir
- étape 3 : reçu une tâche d'indice de priorité 8 à accomplir
- étape 4 : accompli sa tâche prioritaire
- étape 5 : reçu une tâche d'indice de priorité 12 à accomplir
- étape 6 : accompli sa tâche prioritaire
- étape 7 : accompli sa tâche prioritaire
- étape 8 : reçu une tâche d'indice de priorité 15 à accomplir
- étape 9 : reçu une tâche d'indice de priorité 19 à accomplir
- étape 10 : accompli sa tâche prioritaire