25NSIPE4

Exercice 2 (6 points)

Cet exercice porte sur les réseaux, le routage, les graphes et la programmation.

Un aéroport dispose d'un réseau informatique décomposé en différents sous-réseaux :

- Navigation (N) : utilisé principalement par la tour de contrôle ;
- Guichets (G) : utilisé aux guichets dans le hall ;
- Achats (A) : utilisé sur les bornes d'achat placées dans le hall ;
- Sécurité (S) : utilisé aux contrôles de sécurité ;
- Portes (P) : utilisé au niveau des portes d'accès aux avions ;
- Bagages (B) : utilisé par les services qui gèrent le transit des bagages ;
- Commerces (C) : utilisé par tous les commerces.

Le réseau possède l'architecture suivante, où R1, R2, R3, R4, R5, R6 et R7 sont des routeurs :

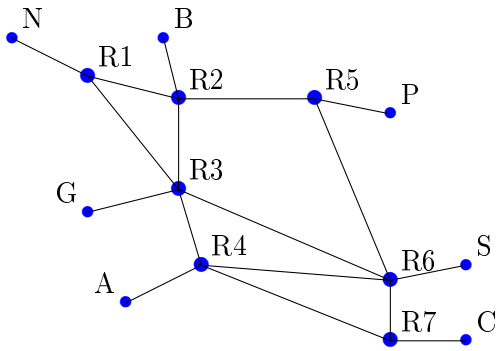


Figure 1. Schéma du réseau.

Partie A : Réseau et adressage

On souhaite ajouter des machines sur le sous-réseau Commerces sur lequel sont déjà connectées 207 machines. L'adresse du sous-réseau est 137.254.128.0 et le masque de sous-réseau utilisé est 255.255.255.0 (l'adresse IP du réseau est donc 137.254.128.0/24 en notation CIDR). On rappelle que cela signifie que les adresses IP du réseau ont toutes en commun leurs 24 premiers bits lorsque les adresses IP sont écrites en binaire.

À part le routeur, toutes les machines déjà présentes sur le sous-réseau sont numérotées dans l'ordre croissant en partant de la plus petite IP disponible.

1. Parmi les deux adresses IP suivantes : 137.254.128.200 et 137.254.128.210, donner l'adresse IP de la machine déjà connectée au sous-réseau Commerces.
2. Préciser s'il est possible ou non d'ajouter 132 machines sur le sous-réseau Commerces, en justifiant la réponse.

Partie B : Programmation d'un protocole de routage

Dans la suite de l'exercice, pour simplifier, on ne considère que les routeurs. Les tables de routage simplifiées sont données dans le tableau suivant, précisant pour chaque routeur en tête de colonne, la passerelle (c'est-à-dire le routeur à contacter) correspondant au routeur destination en début de ligne.

	Source	R1	R2	R3	R4	R5	R6	R7
Destination	R1		R1	R1	R6	R2	R4	R4
	R2	R2		R2	R3	R2	R5	R6
	R3	R3	R3		R3	R6	R3	R4
	R4	R3	R3	R4		R6	R4	R4
	R5	R2	R5	R2	R6		R5	R6
	R6	R2	R5	R6	R6	R6		R6
	R7	R3	R3	R6	R7	R6	R7	

Ainsi, selon ce tableau, si le routeur R3 reçoit des données à transmettre au routeur R5, il enverra ses données au routeur R2.

3. Donner la liste des routeurs par lesquels transite un message envoyé depuis une machine du sous-réseau Navigation à destination d'une machine du sous- réseau Commerces.
4. Décrire le problème rencontré lorsque qu'une machine du sous-réseau Commerces envoie des données à destination d'une machine du sous-réseau Navigation.

Pour éviter ce problème, on veut reconfigurer les routeurs en réécrivant leurs tables de routage à l'aide d'un programme. Pour y parvenir, on modélisera le réseau par un graphe.

Dans toute la suite, les sommets du graphe, qui représenteront les routeurs du réseau, seront décrits par leur nom (type str) et un graphe sera représenté par un dictionnaire associant à chaque sommet la liste des sommets qui lui sont liés par une arête.

Pour la prochaine question uniquement, on considère le réseau obtenu en se limitant aux routeurs R1, R2, R3 et R5. On obtient alors le réseau suivant :

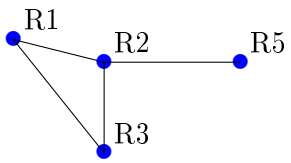


Figure 2. Schéma du réseau restreint aux routeurs R1, R2, R3 et R5.

5. Donner le dictionnaire correspondant au réseau de la Figure 2.
6. Rappeler le principe d'une fonction récursive.

Pour remplir les tables de routage en évitant le problème soulevé à la question 4, on souhaite utiliser le protocole RIP, qui minimise le nombre de routeurs par lesquels les paquets transitent. Une première idée est de construire la liste de tous les chemins possibles reliant ces deux routeurs puis de choisir un chemin le plus court possible dans cette liste.

On suppose que l'on dispose d'une fonction `liste_chemins(graphe, r_depart, r_arrivee)` qui prend en paramètres un graphe, un routeur de départ et un routeur d'arrivée et qui renvoie la liste de tous les chemins liant les deux routeurs, les chemins étant représentés par les listes des routeurs par lesquels passer.

En notant `g` le graphe écrit à la question 5, on a donc :

```
>>> liste_chemins(g, 'R1', 'R5'))
[['R1', 'R2', 'R5'], ['R1', 'R3', 'R2', 'R5']]
```

On a besoin de connaître un chemin le plus court possible entre deux routeurs en utilisant le protocole RIP.

7. Écrire une fonction `plus_court_chemin(graphe, r_depart, r_arrivee)` qui renvoie une liste représentant un des plus courts chemins entre les routeurs `r_depart` et `r_arrivee` en utilisant le protocole RIP. On utilisera la fonction `liste_chemins` définie à la question précédente.

L'agent responsable du réseau consulte un informaticien au sujet de cette fonction. Il lui explique que cette fonction a un défaut : construire tous les chemins liant deux routeurs peut être long pour un réseau étendu. En effet, le nombre de chemins augmente de façon quasi exponentielle avec le nombre de routeurs. Pour remédier à ce problème et améliorer le temps d'exécution de la recherche d'un plus court chemin, l'informaticien lui propose d'utiliser une autre approche basée sur **un parcours en largeur du graphe** (voir Annexe A). En effet, avec un tel parcours, si un chemin est trouvé, il est forcément de longueur minimale.

8. Compléter la fonction `plus_court_chemin_largeur(graphe, r_depart, r_arrivee)` suivante qui traduit l'idée de l'informaticien, réalisant un parcours en largeur et dans laquelle le dictionnaire `dict_chemins` associe à un routeur le chemin reliant `r_depart` à ce routeur.

```
def plus_court_chemin_largeur(graphe, r_depart, r_arrivee):
    dict_chemins = {}
    L = [r_depart]
    sommets_marques = [r_depart]
    dict_chemins[r_depart] = [r_depart]
    for r in L:
        for s_r in graphe[r]:
            if not s_r in sommets_marques:
                sommets_marques.append(...)
                dict_chemins[s_r] = dict_chemins[r] + [s_r]
                if s_r == r_arrivee :
                    return ...
                L.append(s_r)
```

9. Écrire alors une fonction `table_routage(graphe, routeur)` qui renvoie la table de routage du routeur passé en paramètre sous la forme d'un dictionnaire associant à chaque routeur destination la passerelle orrespondante. On pourra utiliser les fonctions écrites dans les questions précédentes.

Partie C : Utilisation du protocole OSPF

Le réseau utilise trois types de connexion :

- Ethernet (E) : débit de 10 megabits par seconde ;
- Fast Ethernet (FE) : débit de 100 megabits par seconde ;
- Fibre (F) : débit de 500 megabits par seconde.

Les types de connexion sont reportés sur la figure du réseau suivante :

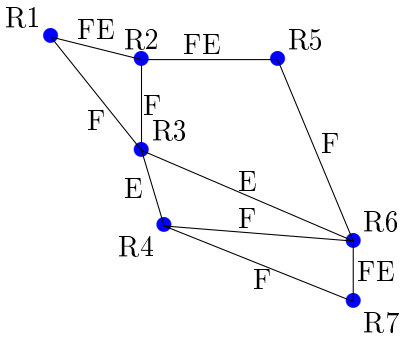


Figure 1. Types de connexions du réseau.

La qualité des liaisons entre les routeurs étant de natures différentes, on décide finalement d'opter pour un routage utilisant le protocole OSPF (Open Shortest Path First). On rappelle que le protocole OSPF configure les routeurs en privilégiant les routes dont le coût total est minimal, où le coût des connexions est donné par la formule suivante : $\text{coût} = \dfrac{10^9}{\text{débit}}$, où le débit est exprimé en bits par seconde.

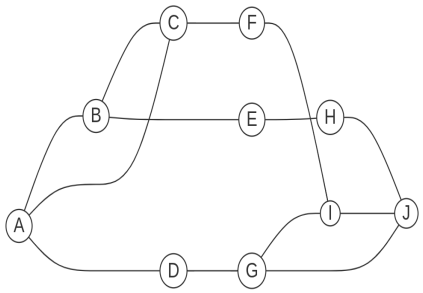
10. Calculer le coût correspondant à chaque type de liaison.
11. Donner la liste des routeurs par lesquels transite un message envoyé depuis le routeur R1 à destination du routeur R7 en respectant le protocole OSPF. *(On pourra utiliser l'algorithme de Dijkstra)*
12. Recopier et compléter la table de routage du routeur R2 toujours en respectant le protocole OSPF.

Destination	Suivant
R1	
R2	
R3	
R4	
R5	
R6	
R7	

A Le Parcours en Largeur d'un graphe

Le Parcours en Largeur (ou BFS pour Breadth-First Search) est un algorithme d'exploration de graphe qui explore systématiquement tous les sommets à une distance k d'un sommet source avant d'explorer les sommets à la distance $k+1$.
Un parcours en largeur débute à partir d'un nœud source. Ensuite, il liste tous les voisins de la source, pour ensuite les explorer un par un. Ce mode de fonctionnement utilise donc une file dans laquelle il prend le premier sommet et place en dernier ses voisins non encore explorés. Les nœuds déjà visités sont marqués afin d'éviter qu'un même nœud soit exploré plusieurs fois (Source : *wikipedia*).

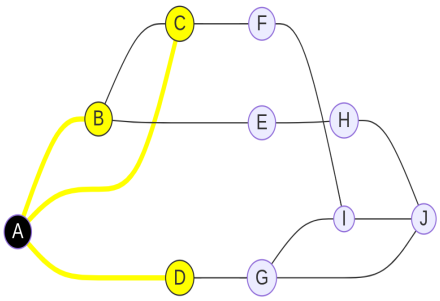
Considérons le graphe suivant :



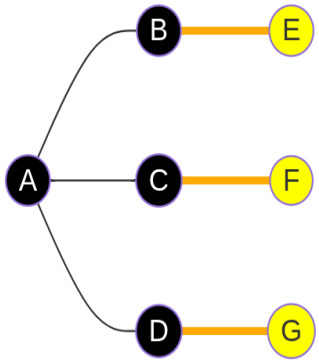
Étape	Sommet traité	File (FIFO)	État de la découverte (Niveaux)
0	(Initialisation)	[A]	**Niveau 0 :** A
1	**A**	[B, C, D]	**Niveau 1 :** B, C, D (voisins directs de A)
2	**B**	[C, D, E]	B est fini, on ajoute son voisin blanc E.
3	**C**	[D, E, F]	C est fini, on ajoute son voisin blanc F.
4	**D**	[E, F, G]	D est fini, on ajoute son voisin blanc G.
5	**E**	[F, G, H]	**Niveau 2 :** E, F, G (tous à distance 2 de A)
6	**F**	[G, H, I]	F est fini, on ajoute I.
7	**G**	[H, I, J]	G est fini, on ajoute J.
8	**H**	[I, J]	H voit J (déjà gris), il ne l'ajoute pas.
9	**I**	[J]	**Niveau 3 :** H, I, J
10	**J**	[]	Fin du parcours.

TABLE 1 – Déroulement du parcours en largeur (BFS)

Prenons le sommet A comme source, ils possèdent trois voisins directs B, C, D (distance $k = 1$) :



Ensuite on défile (ici B car en tête de file) et on enfile tous les voisins non encore rencontrés de ce noeud (c'est le cas du sommet E). On construit ainsi un arbre des chemins depuis la source :



L'algorithme de Dijkstra peut être vu comme une généralisation du parcours en largeur (avec des arcs pondérés positivement). Dans notre exemple, on obtient l'arbre final :