

Devoir Surveillé 1

*Durée : 2 heures
Calculatrice interdite*

Exercice 0.1. Nous avons à notre disposition la base de données d'une station de ski composée de tables dont la structure est la suivante :

<i>Domaines</i>	
<i>id_domaine</i>	<i>nom</i>

La table « *Domaines* » contient les informations sur les domaines skiables pour lesquels un forfait peut être acheté. Par exemple, un nom de domaine possible est 'Débutant'.

<i>Bornes</i>				
<i>id</i>	<i>id_borne</i>	<i>nom</i>	<i>id_domaine</i>	<i>temps</i>

La table "Bornes" contient les informations utiles de chaque borne permettant l'accès à une remontée mécanique, l'information importante étant de quel domaine elle fait partie. Une même borne peut appartenir à plusieurs domaines. Le temps correspond au temps minimum nécessaire pour la remontée concernée.

<i>Bureaux</i>		
<i>id_bureau</i>	<i>nom</i>	<i>type</i>

La table « *Bureaux* » contient les noms des différents bureaux où peuvent être achetés des forfaits, et leurs types (bornes automatiques, caisse).

<i>Forfaits</i>					
<i>id_forfait</i>	<i>id_domaine</i>	<i>id_bureau</i>	<i>date</i>	<i>heure</i>	<i>validité</i>

La table "Forfaits" contient tous les forfaits achetés, domaine de validité, bureau d'achat, date et heure de début de validité et durée de validité en heures. Tous les forfaits sont initialisés pour un début de validité à 00h00. Pour les forfaits 5h, l'heure sera mise à jour lors du premier passage dans une borne.

<i>Clients</i>			
<i>id_client</i>	<i>nom</i>	<i>prenom</i>	<i>id_forfait</i>

La table « *Clients* » contient pour chaque client, son nom et son prénom, et l'id du forfait qu'il a acheté.

<i>Passages</i>					
<i>id_passage</i>	<i>id_borne</i>	<i>date</i>	<i>heure</i>	<i>id_forfait</i>	<i>passage</i>

La table « Passages » répertorie, à chaque tentative de passer à une borne, l'id de la borne, la date et heure de passage du forfait, l'id du forfait associé et le résultat (passage autorisé (1) ou non(0)).

La base de données relationnelle est donc :

- Domaines(id domaine : Integer, nom : Text)
- Bornes(id : Integer, id_borne : Integer, nom : Text, id_domaine : Integer, Temps : Integer)
- Bureaux(id bureau : Integer, nom : Text, type : Text)
- Forfaits(id_forfait : Integer, id_domaine : Integer, id_bureau : Integer, date : Date, heure : Datetime, validité : Integer)
- Clients(id_client : Integer, nom : Text, prenom : Text, id_forfait : Integer)
- Passages(id_passage : Integer, id_borne : Integer, date : Date, heure : Datetime, id_forfait : Integer, passage : Integer)

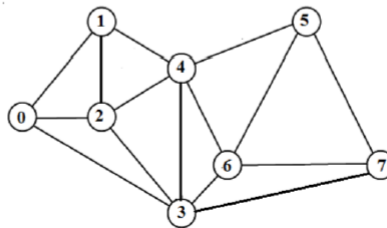
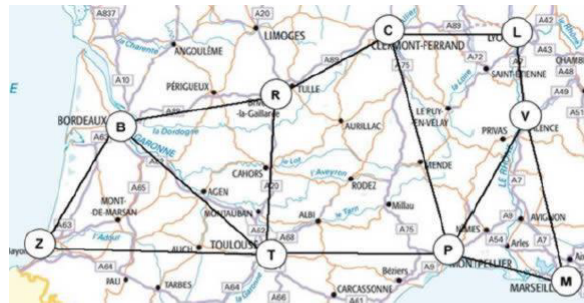
Rédiger les requêtes SQL suivantes :

1. Lister les noms des bornes sans doublons
2. Afficher le nombre de domaines.
3. Trouver le temps minimum de remontée sur l'ensemble des bornes.
4. Trouver les id des passages autorisés du forfait 4.
5. Trouver la date à laquelle la station est passée en numérique : premier passage de forfait.
6. Trouver les deux premières dates de passage du forfait 4.
7. Donner les couples nom/prénom des personnes ayant acheté un forfait le 31/12/2020.
8. Trouver la liste des heures distinctes de passage de forfaits au « TK Chavanette ».
9. Lister date et heure des deux derniers passages de la cliente « Laure Topédie » dans la station, quel que soit son forfait.

Exercice 0.2. Un réseau routier peut être représenté par un dessin qui se compose de points et de traits continus reliant deux à deux certains de ces points : les points sont les villes, et les lignes sont les routes.

On considère que toutes les routes sont à double sens. Chaque route peut être affectée par une valeur qui peut représenter le temps ou la distance entre deux villes, ...

Par exemple, ci-dessous, un réseau routier composé de **15** routes, et de **8** villes numérotées de **0** à **7**.



Partie 1 - Modélisation d'un réseau routier

On considère un réseau routier composé de n villes (avec $n \geq 2$). Les villes du réseau routier sont numérotées par des entiers allant de 0 à $n - 1$.

Pour plus de clarté, tous les exemples de ce problème seront appliqués sur le réseau routier de la figure ci-dessus.

Nota : pour l'ensemble du sujet, on privilégiera le réemploi des fonctions créées.

I. 1- Représentation du réseau routier en Python

Pour représenter les liaisons entre villes, on utilise une matrice d'adjacence *Reseau* d'ordre n , composée d'une liste composée de n listes de même longueur n . On utilise '1' si les villes sont reliées, '0' sinon.

1. Écrire les 4 premières lignes de cette matrice *Reseau*.
Indiquer sa particularité en la justifiant.
2. Donner les résultats et la signification des 4 expressions suivantes :
`Reseau[1]`, `Reseau[4][2]`, `len(Reseau)`, `len(Reseau[2])`
3. Proposer le code permettant de déclarer ce réseau sous forme d'un dictionnaire *Reseau_dic*.
On pourra se limiter aux 3 premiers termes.

I. 2- Villes voisines

Les villes i et j sont dites voisines, lorsqu'il existe une route entre la ville i et la ville j .

4. Écrire la fonction **voisines** (*i*, *j*, **R**), qui reçoit en paramètres deux villes *i* et *j* d'un réseau routier représenté par la matrice symétrique **R**.
La fonction renvoie *True* si les villes *i* et *j* sont voisines, *False* sinon.
5. Écrire la fonction **list_voisines** (*i*, **R**) renvoyant la liste de toutes les villes voisines à la ville *i*.

I. 3- Degré des villes

Dans un réseau routier, le **degré** d'une ville *i* est le nombre de villes voisines à la ville *i*.

6. Écrire la fonction **degre_villes** (**R**) renvoyant une liste *D* contenant des tuples.
Chaque tuple de *D* est composé de deux éléments : une ville du réseau routier, et son degré.

On propose la fonction **X_degrees**(*D*), ci-dessous :

```
def X_degrees (D):
    for i in range (len(D) -1):
        min_idx = i
        for j in range (i+1, len(D)):
            if D[ min_idx ][1] < D[j ][1]:
                min_idx = j
            D[i], D[ min_idx ] = D[ min_idx ], D[i]
    res = []
    for elm in D:
        res.append (elm [0])
    return res
```

D étant la liste de tuples précédente :

$$D = [(0, 3), (1, 3), (2, 4), (3, 5), \text{etc...}]$$

7. Écrire les états successifs de la liste *D* pour $i = 0, i = 1, i = 2$ et $i = 3$ (en fin de boucle, ligne 7) et le résultat retourné par la fonction.
Indiquer alors le rôle et le nom précis de cette fonction.

Partie 2 - Coloration optimale des villes d'un réseau routier

Une coloration des villes du réseau routier est une affectation de couleurs à chaque ville, de façon à ce que deux villes voisines soient affectées par deux couleurs différentes.

On cherche à construire une liste C qui contiendra les couleurs des villes.

Ces couleurs seront représentées par des entiers strictement positifs $0, 1, 2, 3, \dots$: chaque élément $C[k]$ contiendra la couleur de la ville k du réseau routier.

Pour construire la liste C des couleurs, on propose d'utiliser un algorithme, appelé : *algorithme glouton*.

C'est un algorithme couramment utilisé dans la résolution de ce genre de problèmes, afin d'obtenir des solutions optimales.

Principe de l'algorithme de glouton :

- V est la liste des villes triées dans l'ordre décroissant des degrés des villes
- C est une liste de même taille que V , initialisée par des 0
- Pour toute ville k de V : $C[k]$ est la première couleur non utilisée par les villes voisines à la ville k
- Retourner la liste C

II. 6- Premier entier

8. Écrire la fonction `premier_entier(L)`, qui reçoit en paramètre une liste L de nombres entiers positifs.

La fonction renvoie le premier entier positif qui n'appartient pas à la liste L .

Exemples : `premier_entier([0, 0, 3, 1, 0, 1, 0, 0])` renvoie 2.

`premier_entier([0, 1, 3, 5, 2, 4])` renvoie 6.

II. 7- Liste des couleurs des villes voisines à une ville

9. Écrire la fonction `couleurs_voisines(k, C, R)`, qui reçoit en paramètres une ville k d'un réseau routier représenté par la matrice symétrique R , et la liste C des couleurs des villes du réseau.

La fonction renvoie la liste des $C[i]$ telle que i est une ville voisine à la ville k .

Exemples : `couleurs_voisines(0, [0, 0, 3, 1, 2, 0, 3, 0], Reseau)` renvoie `[0, 3, 1]`

II. 8- Coloration des villes

10. Écrire la fonction `couleurs_villes(R)`, qui reçoit en paramètre la matrice symétrique `R` représentant un réseau routier.
La fonction renvoie la liste `C` des couleurs des villes, en utilisant le principe de glouton cité ci-dessus.

Exemple : `couleurs_villes(Reseau)` renvoie

[2, 1, 3, 1, 2, 1, 3, 2]