SUJET 0 – 2024 – sujet1 - Correction

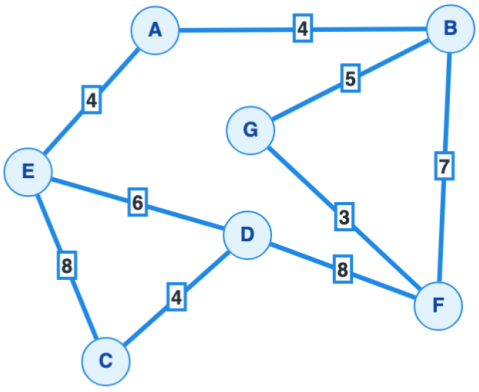
Exercice 3 (8 points)

Partie Graphes

1. Représenter les villes sous forme de graphe pondéré G1

On construit un **graphe non orienté pondéré** :

Sommets : A, B, C, D, E, F, G



Arêtes pondérées :

- A — B (4)
- A — E (4)
- B — F (7)
- B — G (5)
- C — E (8)
- C — D (4)
- D — E (6)
- D — F (8)
- F — G (3)

👉 **Commentaire :** *Il fallait représenter un graphe non orienté car les routes sont à double sens. Chaque arête porte un poids correspondant à la distance.*

---

2. Chemin le plus court entre A et D

On applique un raisonnement type Dijkstra.

Chemins possibles :

- A → E → D = 4 + 6 = 10
- A → B → F → D = 4 + 7 + 8 = 19
- A → B → G → F → D = 4 + 5 + 3 + 8 = 20

- A → E → C → D = 4 + 8 + 4 = 16

Le plus court est : **A → E → D = 10 km**

👉 **Commentaire :** *Le chemin minimal passe par E. Il fallait comparer les longueurs totales et non le nombre de sommets.*

---

3. Matrice d'adjacence de G1

Ordre alphabétique : A, B, C, D, E, F, G. (0 si pas d'arête)

	A	B	C	D	E	F	G
A	0	4	0	0	4	0	0
B	4	0	0	0	0	7	5
C	0	0	0	4	8	0	0
D	0	0	4	0	6	8	0
E	4	0	8	6	0	0	0
F	0	7	0	8	0	0	3
G	0	5	0	0	0	3	0

👉 **Commentaire :** *La matrice est symétrique car le graphe est non orienté.*

---

4. Implémentation de G2 en Python

```python
G2 = {
    'A': ['B', 'C', 'H'],
    'B': ['A', 'I'],
    'C': ['A', 'D', 'E'],
    'D': ['E','C'],
    'E': ['C', 'D', 'G'],
    'F': ['G', 'I'],
    'G': ['E', 'F', 'H'],
    'H': ['A', 'G', 'I'],
    'I': ['B', 'F', 'H']
}
```

👉 **Commentaire :** *Un dictionnaire est adapté : chaque clé est un sommet, la valeur associée est la liste de ses voisins.*

---

5. Parcours en largeur depuis A

Ordre possible : A → B → C → H → I → E → G → F → D

👉 **Commentaire :** *Le parcours en largeur explore niveau par niveau en utilisant une file.*

Partie Programme Python

6. Pourquoi cherche_itineraires est récursive ?

La fonction s'appelle elle-même :

```
nchemin = cherche_itineraires(G, u, end, chaine)
```

👉 **Commentaire :** *Une fonction est récursive lorsqu'elle s'appelle elle-même. Ici, elle explore les chemins possibles en profondeur.*

---

7. Rôle de cherche_itineraires

Elle explore **tous les chemins possibles** entre start et end et les stocke dans la liste globale `tab_itineraires`.

👉 **Commentaire :** *C'est une exploration exhaustive des chemins (type DFS).*

---

8. Compléter itineraires_court

```python
def itineraires_court(G, dep, arr):
    global tab_itineraires
    tab_itineraires = []
    cherche_itineraires(G, dep, arr)
    tab_court = []
    mini = float('inf')
    for v in tab_itineraires:
        if len(v) <= mini:
            mini = len(v)
    for v in tab_itineraires:
        if len(v) == mini:
            tab_court.append(v)
    return tab_court
```

👉 **Commentaire :** *On cherche d'abord la longueur minimale, puis on sélectionne les chemins ayant cette longueur.*

9. Explication du bug

Le problème vient de la variable globale :

```
tab_itineraires = []
```

Elle n'est **pas réinitialisée** entre deux appels, donc les anciens chemins restent mémorisés.

👉 **Commentaire :** *L'utilisation d'une variable globale est dangereuse. Il fallait la réinitialiser dans la fonction.*

Partie Bases de données

---

10. Pourquoi utiliser un SGBD ?

- gestion sécurisée des données
- requêtes rapides (SQL)
- gestion des relations
- cohérence et contraintes

👉 **Commentaire :** *Un fichier texte ne permet ni requêtes complexes ni intégrité référentielle.*

---

11. Schéma relationnel de ville

```
ville(id, nom, num_dep, nombre_hab, superficie)
```

👉 **Commentaire :** *id est la clé primaire.*

---

12. Rôle de id_ville dans sport

C'est une **clé étrangère** vers ville(id).

👉 **Commentaire :** *Elle permet de relier chaque infrastructure à sa ville.*

---

13. Résultat de la requête SQL

```sql
SELECT nom
FROM ville
WHERE num_dep = 74 AND superficie > 70
```

Département 74 :

- Annecy (67 km² ❌)
- Chamonix (246 km² ✅)

Résultat : **Chamonix**

👉 **Commentaire :** *Il fallait vérifier les deux conditions avec AND.*

---

14. Lister les piscines

```sql
SELECT nom
FROM sport
WHERE type = 'piscine';
```

👉 **Commentaire :** *Filtrage simple avec WHERE.*

---

15. Modifier la note

```sql
UPDATE sport
SET note = 7
WHERE nom = 'Ballons perdus' AND note = 6;
```

👉 **Commentaire :** *On sécurise la modification avec la condition sur la note.*

---

16. Ajouter Toulouse

```sql
INSERT INTO ville
VALUES (8, 'Toulouse', 31, 471941, 118);
```

👉 **Commentaire :** *Il faut respecter l'ordre des attributs.*

---

17. Murs d’escalade à Annecy

```sql
SELECT sport.nom
FROM sport
JOIN ville ON sport.id_ville = ville.id
WHERE ville.nom = 'Annecy'
AND sport.type = 'mur d’escalade';
```

Résultat : Aucun (Annecy possède seulement "Ballons perdus", terrain multisport).

👉 **Commentaire :** *Une jointure est nécessaire pour croiser les deux tables.*
