

Devoir surveillé n°7 – NSI
04/03/2025**Exercice 1**

Cet exercice porte sur les graphes et les algorithmes sur les graphes.

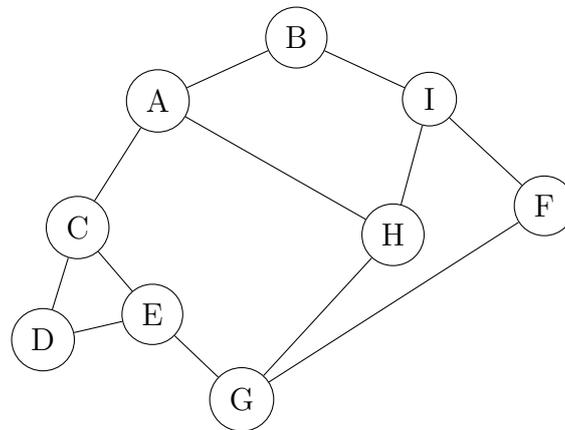
La société CarteMap développe une application de cartographie-GPS qui permettra aux automobilistes de définir un itinéraire et d'être guidés sur cet itinéraire. Dans le cadre du développement d'un prototype, la société CarteMap décide d'utiliser une carte fictive simplifiée comportant uniquement 7 villes : A, B, C, D, E, F et G et 9 routes (toutes les routes sont considérées à double sens).

Voici les routes :

- A est relié à B (4 km de distance);
- A est relié à E (4 km de distance);
- B est relié à F (7 km de distance);
- B est relié à G (5 km de distance);
- C est relié à E (8 km de distance);
- C est relié à D (4 km de distance);
- D est relié à E (6 km de distance);
- D est relié à F (8 km de distance);
- F est relié à G (3 km de distance).

1. Représenter ces villes et ces routes sur sa copie en utilisant un graphe pondéré, nommé G1.
2. Déterminer le chemin le plus court possible entre les villes A et D.
3. Définir la matrice d'adjacence du graphe G1 (avec les sommets triés par ordre alphabétique).

Dans la suite de l'exercice, on ne tiendra plus compte de la distance entre les différentes villes et le graphe G2, non pondéré et représenté ci-dessous, sera utilisé :



Chaque sommet est une ville, chaque arête est une route qui relie deux villes.

4. Proposer une implémentation en Python du graphe G2 à l'aide d'un dictionnaire.
5. Proposer un parcours en largeur du graphe G2 en partant de A.

La société CarteMap décide d'implémenter la recherche des itinéraires permettant de traverser le moins de villes possible. Par exemple, dans le cas du graphe G2, pour aller de A à E, l'itinéraire A-C-E permet de traverser une seule ville (la ville C), alors que l'itinéraire A-H-G-E oblige l'automobiliste à traverser 2 villes (H et G).

Un programme Python donné ci-après a donc été développé (programme p1).

La fonction `itineraires_court` prend en paramètre un graphe `G`, un sommet de départ `dep` et un sommet d'arrivée `arr`. Cette fonction renvoie une liste Python contenant tous les itinéraires pour aller de `dep` à `arr` en passant par le moins de villes possible.

Exemple (avec le graphe G2) :

```
>>> itineraires_court(G2, 'A', 'F')
[['A', 'B', 'I', 'F'], ['A', 'H', 'G', 'F'], ['A', 'H', 'I', 'F']]
```

```

tab_itineraires=[]
def recherche_itineraires(G, start, end, chaine=[]):
    chaine = chaine + [start]
    if start == end:
        return chaine
    for u in G[start]:
        if u not in chaine:
            nchemin = recherche_itineraires(G, u, end, chaine)
            if len(nchemin) != 0:
                tab_itineraires.append(nchemin)
    return []
def itineraires_court(G,dep,arr):
    recherche_itineraires(G, dep, arr)
    tab_court = ...
    mini = float('inf')
    for v in tab_itineraires:
        if len(v) <= ... :
            mini = ...
    for v in tab_itineraires:
        if len(v) == mini:
            tab_court.append(...)
    return tab_court

```

On rappelle les points suivants :

- la méthode `append` ajoute un élément à une liste Python; par exemple, `tab.append(e1)` permet d'ajouter l'élément `e1` à la liste Python `tab`;
 - en Python, l'expression `['a'] + ['b']` vaut `['a', 'b']`;
 - en python `float('inf')` correspond à l'infini.
6. Expliquer pourquoi la fonction `recherche_itineraires` peut être qualifiée de fonction récursive.
 7. Expliquer le rôle de la fonction `recherche_itineraires` dans le programme `p1`.
 8. Compléter la fonction `itineraires_court`.

Les ingénieurs sont confrontés à un problème lors du test du programme `p1`. Voici les résultats obtenus en testant dans la console la fonction `itineraires_court` deux fois de suite (sans exécuter le programme entre les deux appels à la fonction `itineraires_court`) :

```

= RESTART: /home/user/Documents/p1.py
>>> itineraires_court(G2, 'A', 'E')
[['A', 'C', 'E']]
>>> itineraires_court(G2, 'A', 'F')
[['A', 'C', 'E']]

```

alors que dans le cas où le programme `p1` est de nouveau exécuté entre les 2 appels à la fonction `itineraires_court`, on obtient des résultats corrects :

```

= RESTART: /home/user/Documents/p1.py
>>> itineraires_court(G2, 'A', 'E')
[['A', 'C', 'E']]
>>>
= RESTART: /home/user/Documents/p1.py
>>> itineraires_court(G2, 'A', 'F')
[['A', 'B', 'I', 'F'], ['A', 'H', 'G', 'F'], ['A', 'H', 'I', 'F']]

```

9. Donner une explication au problème décrit ci-dessus. Vous pourrez vous appuyer sur les tests donnés précédemment.