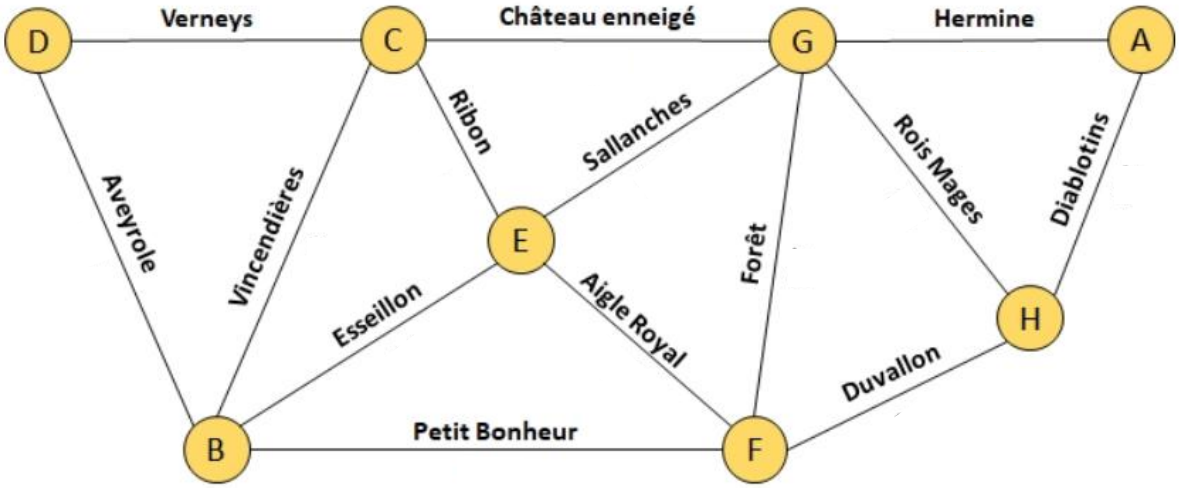
Ds - Graphes

La direction de la station de ski Le Lièvre Blanc, spécialisée dans la pratique du ski de fond, souhaite disposer d'un logiciel lui permettant de gérer au mieux son domaine skiable. Elle confie à un développeur informatique la mission de concevoir ce logiciel.

Le plan des pistes du domaine Le Lièvre Blanc peut être représenté par le graphe ci-dessous. Les sommets correspondent à des postes de secours. Sur chaque arête, on a indiqué le nom de la piste.



1- IMPLEMENTATION PAR LISTE D'ADJACENCE :

Le graphe précédent est implémenté par la liste d'adjacence donnée ci-contre :

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

On dispose d'une classe *File* qui contient les méthodes *enfiler()*, *defiler()* et *estVide()*.

On donne la fonction suivante :

```
def parcoursFile(G,noeudDepart):
    listeVisite = []
    f = File()
    f.enfiler(noeudDepart)
    while not f.estVide() :
        nd = f.defiler()
        if nd not in listeVisite :
            listeVisite.append(nd)
            for voisin in G[nd] :
                if voisin not in listeVisite :
                    f.enfiler(voisin)
    return listeVisite
```

Question 1. : On exécute `parcoursFile(domaine,'A')`. Compléter ci-dessous le contenu le contenu de la file f à la fin des 5 premières itérations générées par la boucle while.

Etat de la file f	*Etat de la file f*
Contenu de f au lancement de la boucle	A
Contenu de f à la fin de la 1ère itération	H G
Contenu de f à la fin de la 2ième itération	H F E C H
Contenu de f à la fin de la 3ième itération	F H F E C
Contenu de f à la fin de la 4ième itération	E D B F H F E
Contenu de f à la fin de la 5ième itération	F B E D B F H F

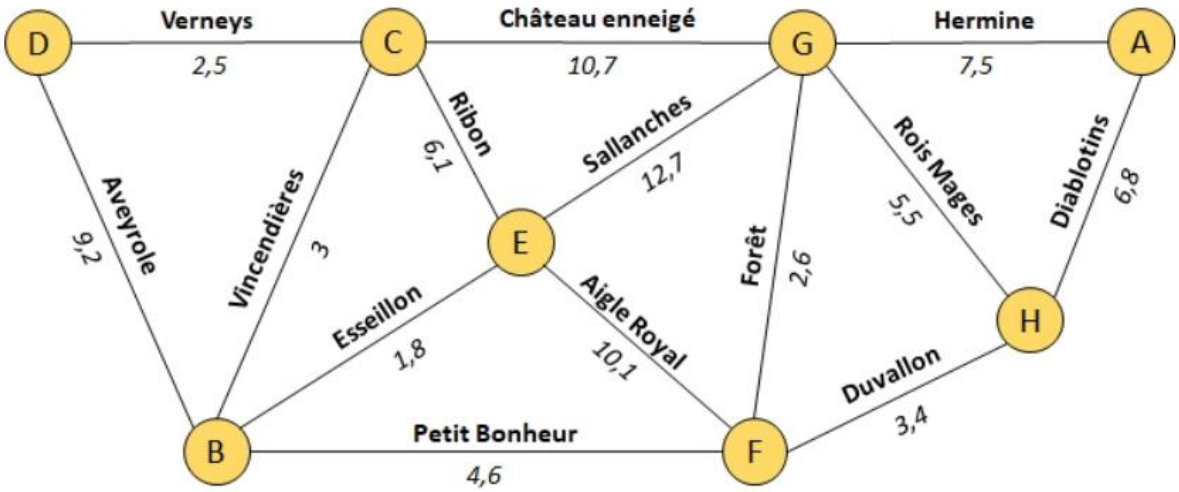
Question 2. : Donner le contenu de la liste retournée.

```
>>> parcoursFile(domaine,'A')
['A', 'G', 'H', 'C', 'E', 'F', 'B', 'D']
```

Question 3. : Comment qualifie-t-on ce type de parcours ?

2- EVOLUTION VERS UN GRAPHE PONDERE :

On décide d'améliorer le modèle précédent en travaillant à présent sur le graphe pondéré ci-dessous, qui reprend le graphe précédent et qui donne en plus les longueurs des pistes en kilomètres.



On implémente à présent le graphe ci-dessus grâce au dictionnaire de dictionnaires suivant :

```
domaine = { 'A' : {'G':7.5 ,'H':6.8} ,
            'B' : {'C':3.0,'D':9.2,'E':1.8,'F':4.6} ,
            'C' : {'B':3.0,'D':2.5,'E':6.1,'G':10.7} ,
            'D' : {'B':9.2,'C':2.5} ,
            'E' : {'B':1.8,'C':6.1,'F':10.1,'G':12.7} ,
            'F' : {'B':4.6,'E':10.1,'G':2.6,'H':3.4} ,
            'G' : {'A':7.5,'C':10.7,'E':12.7,'F':2.6,'H':5.5} ,
            'H' : {'A':6.8,'F':3.4,'G':5.5} }
```

Question 4. : Écrire une instruction Python permettant d'afficher la longueur de la piste allant du sommet 'E' au sommet 'F'.

```
>>> domaine['E']['F']
10.1
```

Question 5. : Écrire une fonction *voisins* qui prend en paramètres un graphe G et un sommet s du graphe G et qui renvoie la liste des voisins du sommet s. Par exemple, l'instruction `voisins(domaine,'B')` renvoie la liste `['C', 'D', 'E', 'F']`

```
def voisins(G,s) :
    l = []
    for v in G[s] :
        l.append(v)
    return l
```

Question 6. : **Recopier** complètement et compléter la fonction *longueur_chemin* donnée ci-dessous : cette fonction prend en paramètres un graphe G et un chemin du graphe G sous la forme d'une liste de sommets et renvoie sa longueur en kilomètres.

Par exemple, l'instruction `longueur_chemin(domaine,['B','E','F','H'])` renvoie `15.3` .

```
1 def longueur_chemin(G, chemin):
2     precedent = ...
3     longueur = 0
4     for i in range(1, len(chemin)):
5         longueur = longueur + ...
6         precedent = ...
7     return ...
```

```
def longueur_chemin(G,chemin):
    precedent = chemin[0]
    longueur = 0
    for i in range(1,len(chemin)):
        longueur = longueur + G[precedent][chemin[i]]
        precedent = chemin[i]
    return longueur
```

En cas de difficulté à s'adapter à l'algorithme donné, possible de reprendre le code (mais **A EVITER)**

```
def longueur_chemin(G,chemin):
    longueur = 0
    for i in range(1,len(chemin)):
        nd1 = chemin[i-1]
        nd2 = chemin[i]
        longueur = longueur + G[nd1][nd2]
    return longueur
```

On donne ci-dessous une fonction *parcours* qui renvoie la liste de tous les chemins du graphe G partant du sommet *depart* et parcourant les sommets de façon unique, c'est-à-dire qu'un sommet est atteint au plus une fois dans un chemin. Par exemple, l'appel `parcours(domaine,'A')` renvoie la liste de tous les chemins partant du sommet *A* , dans le graphe *domaine,* sans se soucier, ni de la longueur du chemin, ni du sommet d'arrivée. Ainsi, ['A', 'G', 'C'] est un chemin possible, tout comme ['A', 'G', 'C', 'B', 'E', 'F', 'H'], tout comme ['A', 'H', 'F'] . Pour information la liste renvoyée par `parcours(domaine,'A')` comprend 224 sous-listes.

```
1 def parcours(G, depart, chemin = [], lst_chemins = []):
2     if chemin == []:
3         chemin = [depart]
4     for sommet in voisins(G, depart):
5         if sommet not in chemin:
6             lst_chemins.append(chemin + [sommet])
7             parcours(G, sommet, chemin + [sommet])
8     return lst_chemins
```

Question 7. : Expliquer en quoi la fonction *parcours* est une fonction récursive.
Cette fonction est récursive car elle s'appelle elle-même en ligne 7.

Un appel à cette fonction *parcours* renvoie une liste de chemins dans laquelle figurent des doublons.
Question 8. : **Recopier** et compléter la fonction *parcours_dep_arr* donnée ci-après. Elle renvoie la liste des chemins partant du sommet *depart* et se terminant par le sommet *arrivee* dans le graphe *G* entrés en paramètres. La liste renvoyée ne doit pas comporter de doublons. Attention, plusieurs lignes de code sont nécessaires. Par exemple, l'exécution de

```
for l in parcours_dep_arr(domaine,'D','A') :
    print(l)
```

donne dans la console, l'affichage donné dans l'annexe en fin d'énoncé.

```
1 def parcours_dep_arr(G, depart, arrivee):
2     liste = parcours(G, depart)
3     ...
```

```
def parcours_dep_arr(G, depart, arrivee):
    liste = parcours(G, depart)
    chemins = []
    for l in liste :
        if l[-1] == arrivee and l not in chemins :
            chemins.append(l)
    return chemins
```

Un pisteur-secouriste de permanence au point de secours D est appelé pour une intervention en urgence au point de secours A. La motoneige de la station étant en panne, il ne peut s'y rendre qu'en skis de fond. Il décide de minimiser la distance parcourue et cherche à savoir quel est le meilleur parcours possible.

Question 9. : Recopier et compléter la fonction *plus_court* donnée ci-dessous. La fonction *plus_court* prend en paramètres un graphe G, un sommet de départ *depart* et un sommet d'arrivée *arrivee*. Elle renvoie un tuple constitué :

- d'un des chemins les plus courts sous la forme d'une liste de sommets ,

- de la longueur en kilomètres de celui-ci.

Par exemple, l'exécution de `plus_court(domaine,'D','A')` renvoie le tuple :
`(['D', 'C', 'B', 'F', 'G', 'A'], 20.2)`

On pourra utiliser la fonction *longueur_chemin* étudiée précédemment.

```
1  def plus_court(G, depart, arrivee):
2      liste_chemins = parcours_dep_arr(G, depart, arrivee)
3      chemin_plus_court = ...
```

```
def plus_court(G, depart, arrivee):
    liste_chemins = parcours_dep_arr(G, depart, arrivee)
    chemin_plus_court = liste_chemins[0]
    minimum = longueur_chemin(G, chemin_plus_court)
    for chemin in liste_chemins[1:]:
        longueur = longueur_chemin(G, chemin)
        if longueur < minimum :
            minimum = longueur
            chemin_plus_court = chemin
    return chemin_plus_court,minimum
```

ANNEXE

```
for l in parcours_dep_arr(domaine,'D','A') :
    print(l)
```

```
['D', 'B', 'C', 'E', 'F', 'G', 'A']
['D', 'B', 'C', 'E', 'F', 'G', 'H', 'A']
['D', 'B', 'C', 'E', 'F', 'H', 'A']
['D', 'B', 'C', 'E', 'F', 'H', 'G', 'A']
['D', 'B', 'C', 'E', 'G', 'A']
['D', 'B', 'C', 'E', 'G', 'F', 'H', 'A']
['D', 'B', 'C', 'E', 'G', 'H', 'A']
['D', 'B', 'C', 'G', 'A']
['D', 'B', 'C', 'G', 'E', 'F', 'H', 'A']
['D', 'B', 'C', 'G', 'F', 'H', 'A']
['D', 'B', 'C', 'G', 'H', 'A']
['D', 'B', 'E', 'C', 'G', 'A']
['D', 'B', 'E', 'C', 'G', 'F', 'H', 'A']
['D', 'B', 'E', 'C', 'G', 'H', 'A']
['D', 'B', 'E', 'F', 'G', 'A']
['D', 'B', 'E', 'F', 'G', 'H', 'A']
['D', 'B', 'E', 'F', 'H', 'A']
['D', 'B', 'E', 'F', 'H', 'G', 'A']
['D', 'B', 'E', 'G', 'A']
['D', 'B', 'E', 'G', 'F', 'H', 'A']
['D', 'B', 'E', 'G', 'H', 'A']
['D', 'B', 'F', 'E', 'C', 'G', 'A']
['D', 'B', 'F', 'E', 'C', 'G', 'H', 'A']
['D', 'B', 'F', 'E', 'G', 'A']
['D', 'B', 'F', 'E', 'G', 'H', 'A']
['D', 'B', 'F', 'G', 'A']
['D', 'B', 'F', 'G', 'H', 'A']
['D', 'B', 'F', 'H', 'A']
['D', 'B', 'F', 'H', 'G', 'A']
['D', 'C', 'B', 'E', 'F', 'G', 'A']
['D', 'C', 'B', 'E', 'F', 'G', 'H', 'A']
['D', 'C', 'B', 'E', 'F', 'H', 'A']
['D', 'C', 'B', 'E', 'F', 'H', 'G', 'A']
['D', 'C', 'B', 'E', 'G', 'A']
['D', 'C', 'B', 'E', 'G', 'F', 'H', 'A']
```

```
['D', 'C', 'B', 'F', 'E', 'G', 'A']
['D', 'C', 'B', 'F', 'E', 'G', 'H', 'A']
['D', 'C', 'B', 'F', 'G', 'A']
['D', 'C', 'B', 'F', 'G', 'H', 'A']
['D', 'C', 'B', 'F', 'H', 'A']
['D', 'C', 'B', 'F', 'H', 'G', 'A']
['D', 'C', 'E', 'B', 'F', 'G', 'A']
['D', 'C', 'E', 'B', 'F', 'G', 'H', 'A']
['D', 'C', 'E', 'B', 'F', 'H', 'A']
['D', 'C', 'E', 'B', 'F', 'H', 'G', 'A']
['D', 'C', 'E', 'F', 'G', 'A']
['D', 'C', 'E', 'F', 'G', 'H', 'A']
['D', 'C', 'E', 'F', 'H', 'A']
['D', 'C', 'E', 'F', 'H', 'G', 'A']
['D', 'C', 'E', 'G', 'A']
['D', 'C', 'E', 'G', 'F', 'H', 'A']
['D', 'C', 'E', 'G', 'H', 'A']
['D', 'C', 'G', 'A']
['D', 'C', 'G', 'E', 'B', 'F', 'H', 'A']
['D', 'C', 'G', 'E', 'F', 'H', 'A']
['D', 'C', 'G', 'F', 'H', 'A']
['D', 'C', 'G', 'H', 'A']
```