TRAVAUX PRATIQUES 10

GRAPHES

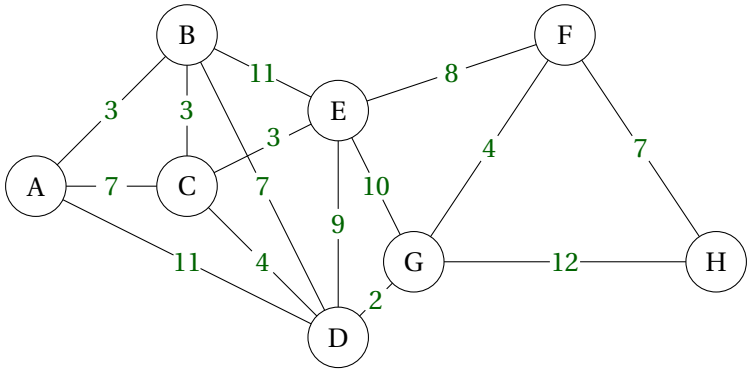
Voici les instructions éventuellement utiles dans ce TP :

`len(L)`	donne la longueur de `L`
`L=[]`	`L` est la liste vide
`L[i]`	donne la valeur de l'élément de la liste d'indice `i`
`L+T`	concatène (juxtapose) les listes `L` et `T`
`L.append(a)`	ajoute `a` à la fin de la liste
`L.insert(i,a)`	ajoute l'élément `a` à la i^e position
`del L[i]`	supprime le i^e élément
`for k in range(i,j)`	boucle **pour** `k` allant **de** `i` (inclus) **à** `j` (exclus)
`while`	boucle **tant que**
`M=array([L1,L2,...])`	`M` est la matrice constituée des lignes `L1`,`L2`...
`len(M)`	donne le nombre de lignes de `M`
`M[i][j]`	donne l'élément situé à la ligne `i` et à la colonne `j`

On peut implémenter une matrice en Python par l'instruction `array` à importer de la bibliothèque `numpy` :

```
from numpy import array
```

On considère la graphe pondéré suivant :



1. Donner une instruction python qui permet de construire la *matrice d'adjacence* $M = (m_{i,j})$ où $m_{i,j}$ est égal à la valeur portée par l'arête reliant les sommets d'indices i et j si ces sommets sont adjacents, ou ∞ sinon.

   Pour cela, on considère la liste des sommets : `S=['A','B','C','D','E','F','G','H']` ; les indices des sommets dans la matrice `M` doivent correspondre aux indices des sommets dans la liste `S`.

   On importera ∞ de la bibliothèque `numpy` par : `from numpy import inf`.

2. Ecrire une fonction python `voisins(M:array,S:list,S0:str)->list`, prenant en arguments la matrice M d'adjacence d'un graphe, la liste S de ses sommets, et un sommet S_0 de ce graphe. Cette fonction renvoie la liste des voisins du sommet S_0.

   Par exemple : `voisins(M,S,'A')` $\longrightarrow$ `['B','C','D']`.

3. Ecrire une fonction `degre(M,S,S0)`, de mêmes arguments que la fonction précédente, qui renvoie le nombre de voisins du sommet S_0 ; c'est-à-dire le nombre d'arêtes issues de S_0.

   Par exemple, `degre(M,S,'A')` $\longrightarrow$ `3`.

4. Écrire une fonction `dico(M:array,S:list)->dict` qui renvoie un dictionnaire dont les clés sont les sommets du graphe et, pour chaque clé, la valeur associée est un tuple constitué du degré du sommet et de la liste de ses voisins.

   Par exemple : `dico(M,S)['A']` $\longrightarrow$ `(3,['B','C','D'])`

5. Pour optimiser l'étude de certaines situations, il est parfois important de trier les sommets par ordre croissant de degré.

   Ecrire alors une fonction `triSommets(M,S)` qui revoie la liste des sommets triée par ordre croissant des degrés.

   Par exemple, `triSommets(M,S)` $\longrightarrow$ `['H','A','F','B','C','G','D','E']`.

6. Ecrire une fonction `longChemin(M,S,L)` qui prend un arguments une matrice M, la liste S de ces sommets et une liste L contenant des sommets (adjacents ou non). Cette fonction renvoie la longueur de ce chemin s'il est réalisable ou ∞ sinon.

   Par exemple, `longChemin(M,S,['A','C','E','F'])` $\longrightarrow$ `18`.

7. Ecrire une fonction `dijkstra(villes,départ,arrivée,matrice)` qui prend en argument une liste (`list`) de villes, le nom (`str`) de la ville de départ, le nom (`str`) de la ville d'arrivée et la matrice (`array`) d'adjacence du graphe ; cette fonction renvoie le plus court chemin entre les deux villes choisies.

   Par exemple : `dijkstra(S,'A','H',M)` $\longrightarrow$ `(23.0,['A','B','D','G','F','H'])`