Graphes : codage

TP 12

Graphes - v1.0

Lycée La Martinière Monplaisir, 41 Rue Antoine Lumière, 69372 Lyon

1 Pays frontaliers en Europe

Dans ce graphe, les pays en Europe sont les sommets et les arêtes correspondent à une frontière terrestre en Europe [1].

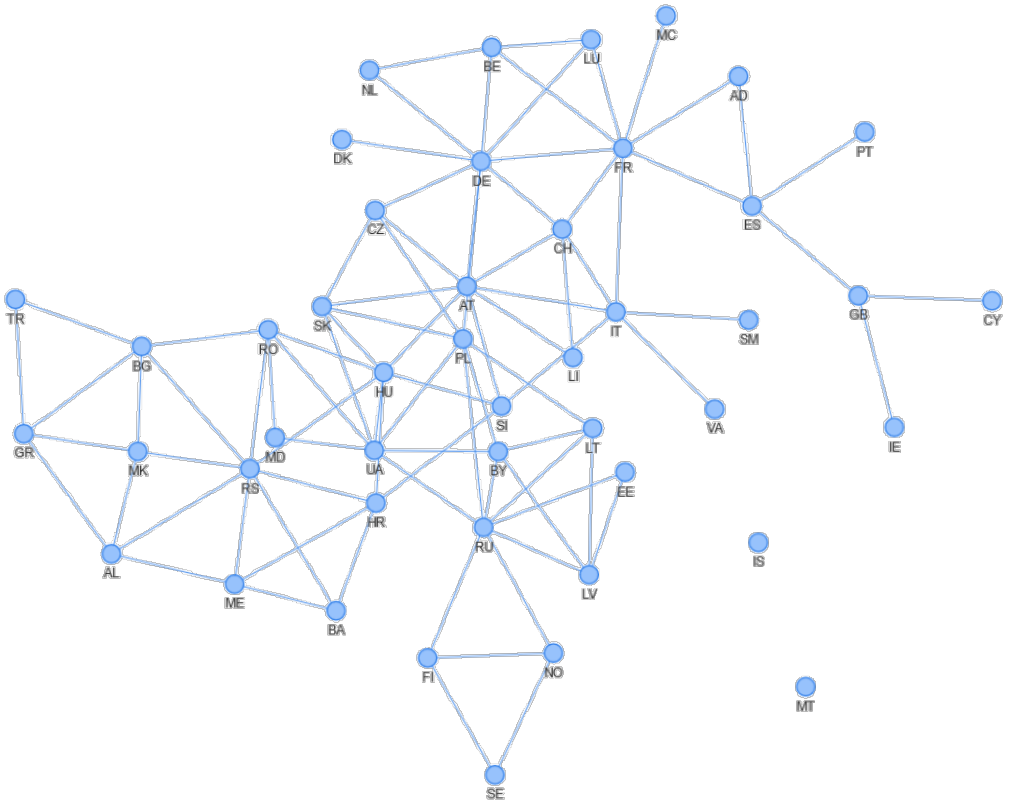

Figure 1 – Les 46 pays en Europe avec leurs 87 frontières terrestres

Un fichier `Europe_etudiants.py` est dans votre espace de classe partagé. Il définit la liste des pays par une liste de 46 chaines de caractères (la chaine de caractère est de longueur 2, car on utilise la norme `ISO 3166-1 alpha-2`) et une liste de listes correspondant aux 87 frontières (l'ordre est alphabétique, la frontière franco-belge est ainsi codée par la liste `['BE', 'FR']`).

1. On n'a donc pas considéré la frontière entre la France et les Pays-Bas à Saint-Martin, mais on a bien considéré la frontière entre la Grande-Bretagne et l'Espagne à Gilbratar et la frontière entre Chypre et les bases britanniques d'Akrotiri et Dhekelia

Il crée également un graphe non orienté en utilisant la bibliothèque `networkx` et le dessine en utilisant `matplotlib`.

Rappel

`L.index(x)` retourne l'indice de l'élément `x` si `x` est dans la liste. C'est une fonction à coût linéaire en la longueur de la liste ($O(n)$).

Pour minimiser la complexité, il peut être préférable d'utiliser un dictionnaire de numérotation `num` tel que `num[p] = i` lorsque `p = pays[i]`.

Question 1 Créer une liste d'adjacence `L` à partir des informations fournies.

Question 2 Vérifier que `L[pays.index("FR")]` renvoie la liste `[1, 2, 5, 11, 19, 23, 27, 42]` (à l'ordre près). Comment obtenir le nom des pays voisins à la France à partir de cette liste ?

Question 3 Écrire une fonction `verifL_GNO(L)` qui vérifie bien que `L` correspond bien à un graphe non orienté (renvoie `True` si c'est le cas, `False` sinon). Cela doit donc vérifier que, si `s2` appartient bien à la liste `L[s1]`, alors `s1` appartient bien à la liste `L[s2]`. Tester votre liste d'adjacence.

Question 4 Créer un dictionnaire d'adjacence `d` à partir des informations fournies.

Question 5 Vérifier que `d["FR"]` renvoie bien `['DE', 'AD', 'BE', 'ES', 'IT', 'LU', 'MC', 'CH']`, ce qui se traduit par : les voisins de la France sont : l'Allemagne, Andorre, la Belgique, l'Espagne, l'Italie, le Luxembourg, Monaco et la Suisse.

Question 6 Écrire une fonction `verifD_GNO(d)` qui vérifie bien que `d` correspond bien à un graphe non orienté. Tester votre dictionnaire d'adjacence.

Question 7 Créer une matrice d'adjacence sous la forme d'un dictionnaire de dictionnaires `dd` tel que `dd[p1][p2]` soit égal à `1` si `p1` possède une frontière commune avec `p2` et `0` sinon.

Question 8 Vérifier que `dd["FR"]["BE"]` retourne bien `1` et que `dd["RU"]["FR"]` retourne bien `0`.

Question 9 Écrire une fonction `verifDD_GNO(dd)` qui vérifie bien que `dd` correspond bien à un graphe non orienté. Tester votre matrice d'adjacence.

2 Vulgarisateurs francophones sur YouTube

Dans ce graphe, des vulgarisateurs francophones à large audience (supérieur à 500 000 abonnés) sont les sommets et les arcs correspondent à une recommandation sur leur page d'accueil [2].

FIGURE 1 – Les 17 chaines YouTube concernées avec leurs 29 arcs

Un fichier `vulga_etudiants.py` est dans votre espace de classe partagé. Il définit la liste des sommets (une liste de chaine de caractères) et une liste d'arcs (liste de listes de chaine de caractères). Il crée également un graphe orienté en utilisant la bibliothèque `networkx` et le dessine en utilisant `matplotlib`.

Question 1 Écrire une fonction `dictAdj(s, a)` qui prend en entrée une liste de sommets `s` et une liste d'arcs `a` et renvoie le dictionnaire d'adjacence correspondant.

Question 2 Utiliser la commande :

2. Relevé le 08/04/2024

`G1 = nx.from_dict_of_lists(dictAdj(sommets, arcs), create_using=nx.DiGraph)`
pour créer un nouveau graphe en utilisant votre fonction. Dessiner ce graphe pour vérifier votre fonction.

Question 3 Quelle est la chaine qui fait le plus de publicité ? Combien recommande-t-elle de collègues ?

Question 4 Écrire une fonction `listeDeListes(s, a)` qui prend en entrée une liste de sommets `s` et une liste d'arcs `a` et renvoie la matrice d'adjacence correspondante sous la forme d'une liste de listes. Il sera utile d'utiliser un dictionnaire de numérotation `num` tel que `num[si] = i` (renvoie l'indice du sommet).

Question 5 Utiliser la commande :
`G2 = nx.from_numpy_matrix(np.array(listeDeListes(sommets, arcs)), create_using=nx.DiGraph)`
pour créer un nouveau graphe en utilisant votre fonction. Dessiner ce graphe pour vérifier votre fonction.

Question 6 Quelle est la chaine qui reçoit le plus de publicité ? Combien de collègues la recommandent ?

Question 7 Écrire une fonction `dictDeDict(s, a)` qui prend en entrée une liste de sommets `s` et une liste d'arcs `a` et renvoie la matrice d'adjacence correspondante sous la forme d'un dictionnaire de dictionnaires `d` : `d[s1][s2]` doit renvoyer `0` si l'arc (s_1, s_2) n'existe pas et `1` si l'arc (s_1, s_2) existe.

3 Déplacement du cavalier sur un échiquier

Un cavalier se déplace, lorsque c'est possible, de 2 cases dans une direction verticale ou horizontale, et de 1 case dans l'autre direction (le trajet dessine une figure en L).

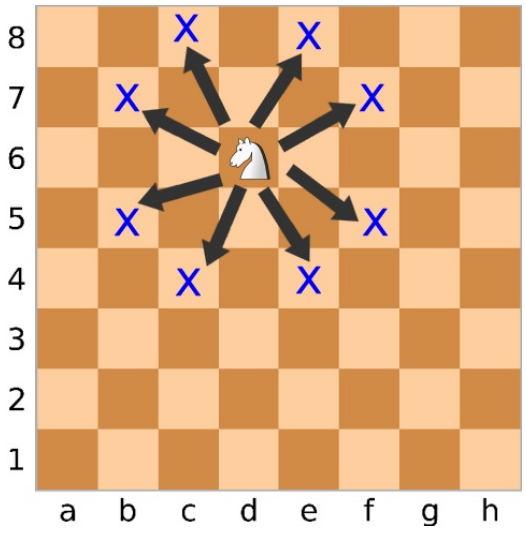

FIGURE 1 – Illustration du mouvement d'un cavalier sur un échiquier

Dans un premier temps, les cases de l'échiquier sont représentées par des tuples : le couple `(i, j)` désigne la case d'abscisse `i` et d'ordonnée `j`. Un échiquier possède 8 colonnes et 8 lignes, donc `i` et `j` seront compris entre `0` et `7`.

Question 1 Écrire une fonction `estDansEch(i, j)` qui renvoie `True` si `(i, j)` correspond à une case valide de l'échiquier et `False` sinon.

Question 2 Écrire une fonction `mvtsPossibles(i, j)` qui renvoie la liste des cases où le cavalier peut se déplacer à partir de la case `(i, j)` à l'ordre après.

Question 3 Vérifier que :

- `mvtsPossibles(0, 0)` renvoie `[(1, 2), (2, 1)]`,
- `mvtsPossibles(3, 5)` renvoie bien `[(1, 4), (1, 6), (2, 3), (2, 7), (4, 3), (4, 7), (5, 4), (5, 6)]`,
- `mvtsPossibles(7, 7)` renvoie bien `[(5, 6), (6, 5)]`.

Tous ces résultats sont à l'ordre près.

Question 4 Créer un graphe `G` sous la forme d'un dictionnaire d'adjacence avec pour sommets les différentes cases de l'échiquier et les arrêtes qui correspondent à un mouvement possible du cavalier.

Question 5 Vérifiez que vous avez un graphe avec 64 sommets et 168 arêtes.

Le codage des cases d'échiquier se fait classiquement par une chaine de caractère comprenant : une lettre minuscule pour l'abscisse (de `a` à `h`) et un chiffre pour l'ordonné (de `1` à `8`). La case en bas à gauche de l'échiquier est donc de code `'a1'`.

Remarque

`ord(c)` retourne le codage Unicode correspondant au caractère `c` et `chr(n)` renvoie le caractère dont le codage Unicode est `n`.

Question 6 Écrire une fonction `codage(i, j)` qui renvoie le code correspondant à la case `(i, j)`.

Question 7 Créer un graphe `G2` avec les cases maintenant nommées d'après leur code.

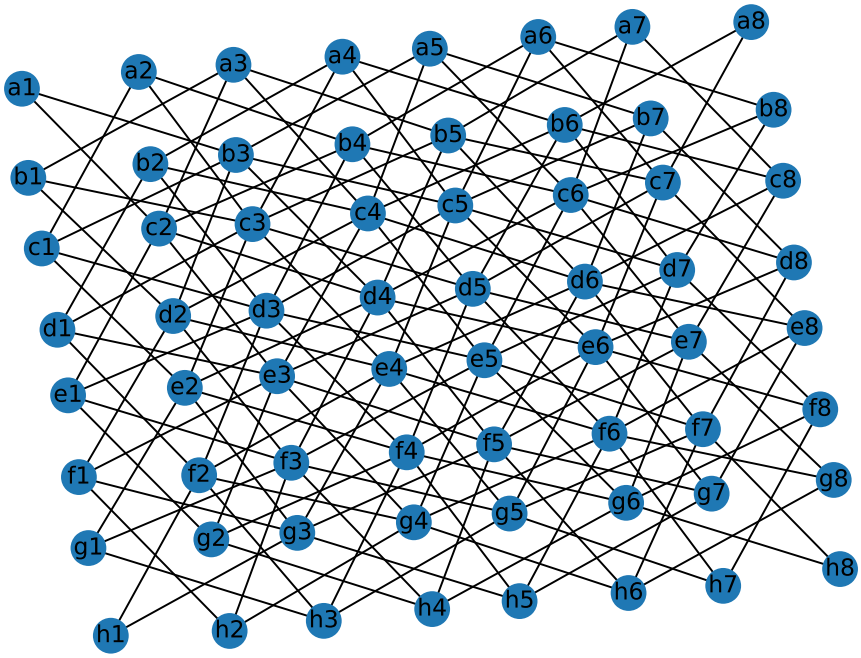

FIGURE 2 – Représentation graphique du graphe `G2`

Question 8 Vérifier que :

- `G2['a1']` renvoie `['b3', 'c2']`,
- `G2['d6']` renvoie bien `['b5', 'b7', 'c4', 'c8', 'e4', 'e8', 'f5', 'f7']`,
- `G2['h8']` renvoie bien `['f7', 'g6']`.

Tous ces résultats sont à l'ordre près.