Leçon d'informatique : Graphes – présentation générale

S. Benlhajlahsen - PCSI$_1$

Sommaire

Extrait du programme :

Notions	Commentaires
Vocabulaire des graphes.	Graphe orienté, graphe non orienté. Sommet (ou nœud) ; arc, arête. Boucle. Degré (entrant et sortant). Chemin d'un sommet à un autre. Cycle. Connexité dans les graphes non orientés. On présente l'implémentation des graphes à l'aide de listes d'adjacence (rassemblées par exemple dans une liste ou dans un dictionnaire) et de matrice d'adjacence. On n'évoque ni multi-arcs ni multi-arêtes.
Notations.	Graphe $G = (S, A)$, degrés $d(s)$ (pour un graphe non orienté), $d_+(s)$ et $d_-(s)$ (pour un graphe orienté).
Pondération d'un graphe. Étiquettes des arcs ou des arêtes d'un graphe.	On motive l'ajout d'information à un graphe par des exemples concrets.

I Présentation

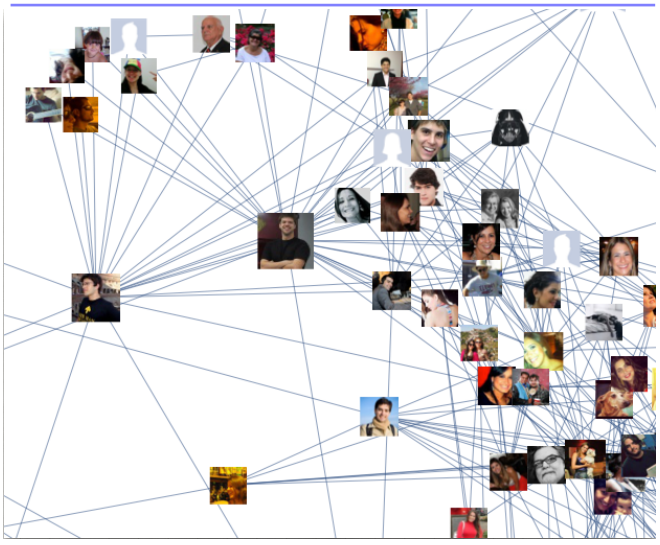

FIGURE 1 – Réseau d'amis sur Facebook

FIGURE 2 – Plan du métro de la ville d'Athènes.

Prenons les figures 1 et 2 qui représentent respectivement un réseau d'amis sur Facebook et le plan du métro de la ville d'Athènes. On voit quelques ressemblances :

- il y a des points appelés dans la suite **sommets** (ou nœud) : ce sont les profils du réseau Facebook ou les station du réseau métropolitain.

- il y a des « traits » qui relient ces sommets : cela signifie une relation d'amitié sur Facebook et, pour le métro, deux stations voisines reliés directement. On parlera dans la suite d'**arête**.

Voici quelques questions que l'on peut se poser :

- **Degré** : Combien d'amis a un profil donné ? Combien de lignes de metro partent depuis une station ?
- **Connexité** : Y a-t-il des profils sans amis sur Facebook ? Existe-t-il toujours un trajet entre deux stations ?

Prenons maintenant le réseau autoroutier. On souhaite aller de Marseille à Paris (3). On voit que, par rapport, au réseau métropolitain, on a rajouté une info : on connait les sommets voisins de Toulouse mais aussi la distance qui les sépare. Les trajets sont alors **pondérés** par les distances. On pourra alors rechercher **le plus court chemin** pour aller à Paris.

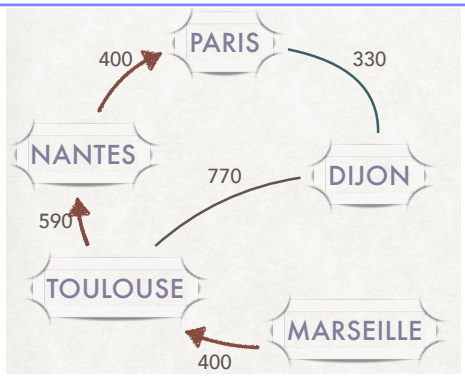

FIGURE 3 – Recherche d'un trajet entre Marseille et Paris.

II Vocabulaire

À retenir : Dans toute la suite, on appellera graphe $G = (S, A)$ un couple formé :

- S d'un ensemble de **sommets** ou **nœuds** ;
- A un ensemble d'**arêtes** (pour les graphes non orientés) ou **arcs** (pour les graphes orientés) qui relient deux sommets.

Exemple : On donne en figures 4 et 5 deux exemples de graphes. Pour le premier graphe :

- $S = \{$"Pierre", "Paul", "Jacques", "Marie", "Alain"$\}$
- $A = \{$("Pierre", "Paul"), ("Paul", "Jacques"), ("Paul", "Marie"), ("Jacques","Marie")$\}$.

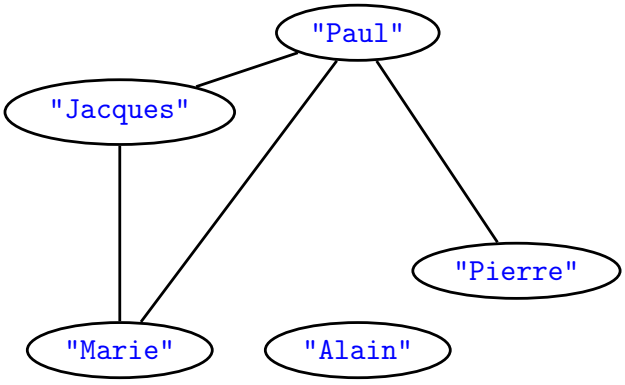

FIGURE 4 – Graphe d'un réseau d'amis sur Facebook.

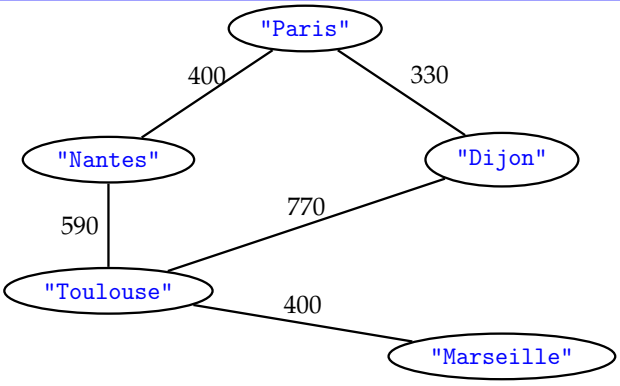

FIGURE 5 – Graphe d'un réseau autoroutier.

Graphe non orienté et graphe orienté : Ces deux derniers graphes sont non orientés. En effet, la relation d'amitié est symétrique et les autoroutes peuvent être parcourues dans les deux sens.
On pourrait aussi imaginer des **graphes orientés** où **les arcs** reliant deux sommets sont représentés par des flèches. Cela

peut représenter une relation asymétrique entre deux sommets :

- relation de subordination dans une structure hiérarchique ;
- trajet entre deux stations qui ne peut se faire que dans un seul sens[a].

a. Exemple des tire-fesses ou remontées mécaniques dans un domaine skiable.

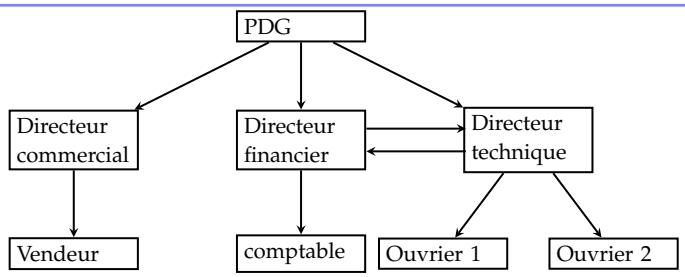

FIGURE 6 – Structure hiérarchique simple d'une entreprise.

Sommets voisins : Dans un graphe non orienté, deux sommets sont dits voisins s'il sont reliés par une arête. Dans le graphe de la figure 4, `"Paul"` a 3 voisins et `"Alain"` n'en a pas.

Degré d'un sommet : Pour un graphe non orienté, le degré d'un sommet est le nombre de voisins de ce sommet, c'est-à-dire le nombre d'arête qui joignent ce sommet. On le note $d(s)$. Pour le graphe de la figure 4, $d(\texttt{"Paul"}) = 3$ et $d(\texttt{"Alain"}) = 0$.
Pour un graphe orienté, on peut définir les degrés sortant $d_+(s)$ et entrant $d_-(s)$:

- le degré sortant $d_+(s)$ d'un sommet s est le nombre d'arc partant de s ;
- le degré entrant $d_-(s)$ d'un sommet s est le nombre d'arc arrivant en s ;

Sur le graphe de la figure 4, le degré sortant du PDG vaut 3 et le degré entrant 0.

Remarque Dans la suite, le mot arête sera réservé aux graphes non orientés et le mot arcs sera réservé aux graphes orientés.

Graphe pondéré (ou valués) : Le graphe de la figure 5 est pondéré car on a attribué un **poids** aux arêtes représentant la distance entre deux villes.
À l'inverse, les graphes des figures 4 et 6 ne sont pas pondérés.

Boucles et multi-arcs :

Tous les graphes précédents étaient **simples**, c'est-à-dire qu'il n'y avait qu'au plus, une seule arête ou arc qui partaient d'un sommet.
Le graphe ci-contre (figure 7) représente la structure des pages d'un site internet. On constate que la page d'accueil `0` permet d'accéder à la page `1`, la page `1` permet d'accéder aux pages `2` et `3`.
De plus, sur la page d'accueil, un lien renvoie vers la même page d'accueil : c'est généralement ce qui se passe quand on clique sur le logo d'un site, cela permet de rafraîchir la page. C'est représenter ici par une `boucle` : un arc qui part d'un sommet pour arriver au même sommet.
De plus, ici, la page `2` contient deux liens qui pointent vers la page `3`. On parle alors de multi-arcs ou de multi-arêtes.

Remarque : Dans toute la suite, on ne parlera plus de multi-arcs ou multi-arêtes.

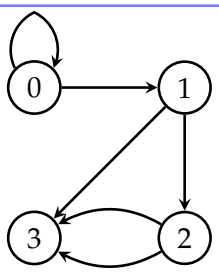

FIGURE 7 – Présence d'une boucle.

Graphe connexe : On notera sur le graphe 4 la présence d'un sommet isolé des autres. Il n'existe aucune arête qui relie `"Alain"` aux autres. On dira alors que ce graphe est **non connexe**

À l'inverse, le graphe de la figure 5, est **connexe** car il est possible, à partir de n'importe quel sommet, de rejoindre tous les autres sommets en suivant les arêtes.
On appellera alors **chemin** un ensemble d'arête qui permettent de relier deux sommets. Un **cycle** est un chemin qui part d'un sommet pour finir au même sommet.
Dans le cas du graphe non connexe de la figure 8, on dira qu'il a 3 **composantes connexes**.

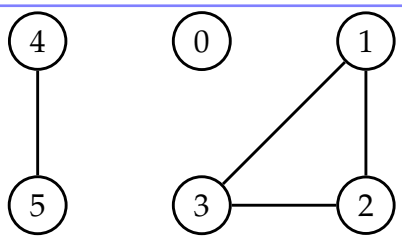

FIGURE 8 – Graphe non connexe

Exercice 1 :

On considère le graphe de la figure 9. Quels sont les degrés sortant $d_+(0)$et entrant $d_-(0)$ du sommet 0 ?

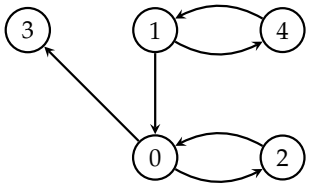

FIGURE 9

Exercice 2 : On considère l'ensemble des entiers $[\![1,6]\!]$. Représenter le graphe de la divisibilité. Autrement dit, la flèche $a \to b$ signifie que a divise b. On excluera les boucles, c'est-à-dire qu'on ne prend pas en compte la division d'un nombre par lui-même.
Réponse : voir figure 10 en page 4.

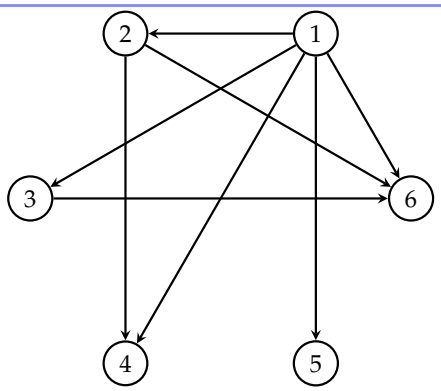

FIGURE 10 – Graphe de divisibilité

III Représentation des graphes en informatique

Pour représenter en Python, des graphes, nous allons proposer plusieurs méthodes. Pour simplifier la suite, on va représenter les sommets par des entiers.

III.A Liste d'adjacence

Cas d'un graphe non pondéré et non orienté :

Prenons le cas du de la figure 12, on va construire pour chaque sommets la liste de ses voisins. Cette liste de liste sera appelée `liste d'adjacence`.
Le résultat est donné en figure 20. La longueur de cette liste vaut le nombre de sommets du graphe.

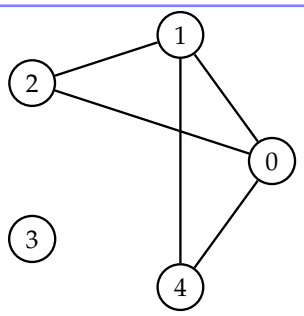

FIGURE 12

```
1 L1 = [ [1, 2, 4],
2        [0, 2, 4],
3        [0, 1],
4        [],
5        [0,1]]
```

FIGURE 11

Cas d'un graphe pondéré et non orienté : Prenons le cas du de la figure 22. Par rapport au cas précédent, on doit, pour chaque voisin, préciser le poids de l'arête. On peut alors définir une liste de liste de couples (voisin, poids). Le résultat est donné en figure 13. La longueur de cette liste vaut le nombre de sommets du graphe.
On constate alors que `L1[i][j]` renvoie le poids de l'arête qui joint `i` à `j`.

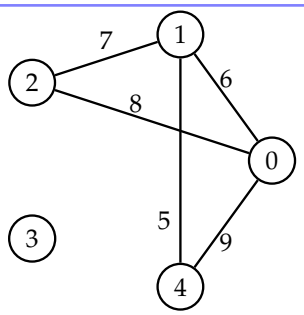

FIGURE 14

```
1 L2 = [[[1, 6], [2, 8], [4, 9]],
2        [[0, 6], [2, 7], [4, 5]],
3        [[0, 8], [1, 7]],
4        [],
5        [[0, 9], [1, 6]]
6       ]
```

FIGURE 13

Cas d'un graphe pondéré et orienté :
Prenons le cas du de la figure 16. Par rapport au cas précédent, il faut tenir compte de l'orientation.
Le résultat est donné en figure 15. La longueur de cette liste vaut le nombre de sommets du graphe.

Remarque : Contrairement au cas du graphe non orienté, `L[i][j]` ne renvoie pas le même résultat que `L[j][i]`.

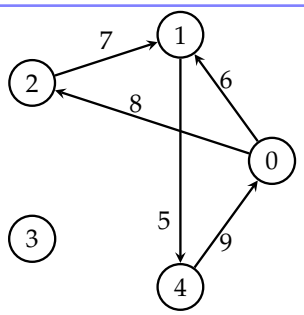

FIGURE 16

```
1 L3 = [[[1, 6], [2, 8]],
2        [[4, 5]],
3        [[1, 7]],
4        [],
5        [[0, 9]]
6       ]
```

FIGURE 15

Exercice 3 : Écrire la liste d'adjacence associée au graphe de la figure 8.
Réponse : voir figure 17 en page 6.

```
L = [[],
     [2, 3],
     [1, 3],
     [1, 2],
     [5],
     [4]
    ]
```

FIGURE 17

III.B Matrice d'adjacence

Soit n le nombre de sommet d'un graphe, on construit la matrice $M = (M_{ij})_{0 \leq i,j \leq n-1}$ telle que m_{ij} représente le poids de l'arête ou de l'arc qui joint i et j.

Cas d'un graphe non pondéré et non orienté :

Dans ce cas , on choisit arbitrairement $m_{ij} = 1$ si deux sommets sont reliés et $m_{ij} = -1$ sinon. Prenons le cas du de la figure 12, la matrice d'adjacence sera :

$$M = \begin{pmatrix} -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 \end{pmatrix}$$

Le résultat est donné en figure 20.

```
M1 = [[-1, 1, 1, -1, 1],
      [1, -1, 1, -1, 1],
      [1, 1, -1, -1, -1],
      [-1, -1, -1, -1, -1],
      [1, 1, -1, -1, -1]
      ]
```

FIGURE 18

Remarque : D'autres conventions existent. On peut prendre $m_{ij} = \infty$ dans le cas où deux sommets ne sont pas reliés.

$$M = \begin{pmatrix} \infty & 1 & 1 & \infty & 1 \\ 1 & \infty & 1 & \infty & 1 \\ 1 & 1 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 1 & 1 & \infty & \infty & \infty \end{pmatrix}$$

Le résultat est donné en figure 19. Le module `math` fournit le flottant `inf` qui simule un infini. En effet, `inf+1` ou `2*inf` renvoie `inf`.

```
from math import inf
M1 = [[inf, 1, 1, inf, 1],
      [1, inf, 1, inf, 1],
      [1, 1, inf, inf, inf],
      [inf, inf, inf, inf, inf],
      [1, 1, inf, inf, inf]]
```

FIGURE 19

Cas d'un graphe pondéré :

Dans le cas d'un graphe pondéré, on prendra, lorsque deux sommets sont voisins, m_{ij} sera égal au poids de l'arête. Ainsi, la matrice d'adjacence du graphe de la figure 22 sera :

$$M = \begin{pmatrix} -1 & 6 & 8 & -1 & 9 \\ 6 & -1 & 7 & -1 & 5 \\ 8 & 7 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ 9 & 5 & -1 & -1 & -1 \end{pmatrix}$$

Le résultat est donné en figure 12.

```
M2 = [[-1, 6, 8, -1, 9],
      [6, -1, 7, -1, 5],
      [8, 7, -1, -1, -1],
      [-1, -1, -1, -1, -1],
      [9, 5, -1, -1, -1]]
```

FIGURE 20

Remarque : Pour un graphe non orienté, la matrice est symétrique $m_{ij} = m_{ji}$.
S'il n'y a pas de boucle, la diagonale sera composée de `-1` ou de ∞ selon les conventions.

III.C Dictionnaire d'adjacence

Les deux premières structures sont très pratiques dans le cas de sommets définies par des sommets entiers dans $[\![0, n-1]\!]$. Dans le cas du réseau d'amis (voir figure 21), on peut d'abord définir la liste des profils auxquels on assiste un entier. Cette méthode amène à stocker deux listes séparées. Une autre solution est l'utilisation d'un dictionnaire.

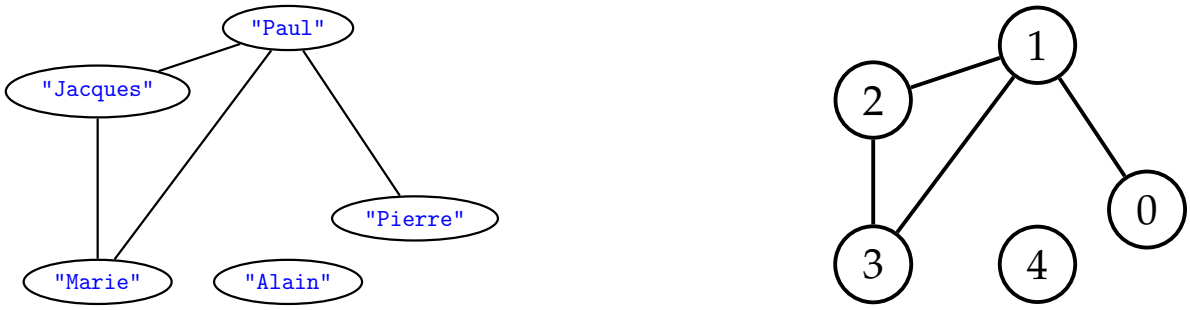

FIGURE 21 – Graphes équivalents

Rappel sur les dictionnaire : Dans un dictionnaire, les éléments sont définis par des valeurs auxquelles on accède par des clés. Par exemple, le dictionnaire `dico = {"pomme":2, 1:9 }`. Pour la valeur 2, la clé est `"pomme"`. On notera les accolades.

Dictionnaire d'adjacence : Dans le cas du graphe de la figure 21, on peut définir un dictionnaire d'adjacence de la même manière que la liste d'adjacence. Pour chaque clé (sommet), la valeur associé est la liste des voisins.

```
dico = {} # dictionnaire vide
dico["Alain"] = [] # alain n'a pas d'amis
dico["Jacques"] = ["Marie", "Paul", "Pierre"]
dico["Marie"] = ["Jacques", "Paul"]
dico["Paul"] =  ["Jacques", "Marie","Pierre"]
dico["Pierre"] = ["Paul"]
print(dico)
```

```
dico = {0: [1], 1: [2, 3, 0], 2: [3, 1, 0], 3: [2, 1], 4: []}
```

Cas d'un graphe pondéré : Comme précédemment, on doit rajouter l'information du poids. Dans le cas du graphe de la figure 22, on donne le dictionnaire d'adjacence. On notera que `dico2[i][j]` renvoie le poids de l'arête.

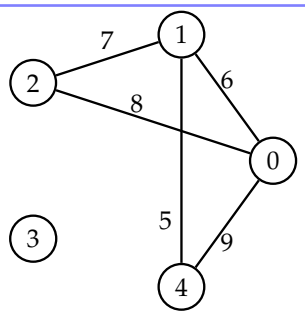

FIGURE 22

```
dico2 = {0:[[1, 6], [2, 8], [4, 9]],
         1:[[0, 6], [2, 7], [4, 5]],
         2:[[0, 8], [1, 7]],
         3:[],
         4:[[0, 9], [1, 6]]
         }
```

Exercice 4 : Écrire le dictionnaire d'adjacence du graphe de la figure 8.
Réponse : voir figure 23 en page 8.

```
dico = {0:[],
        1:[2,3],
        2:[1,3],
        3:[1,2],
        4:[5],
        5:[4],
        }
```

FIGURE 23

IV Quelques exercices de manipulation des graphes

Dans la suite, on manipule uniquement des graphes non orientés et non pondérés à n sommets. Une matrice d'adjacence sera notée par `M`, une liste d'adjacence `L` et un dictionnaire d'adjacence par `D`. Enfin, pour la matrice d'adjacence, on indiquera par -1 si deux sommets ne sont pas reliés et par 1 s'ils le sont.

Exercice 5 : Écrire une fonction `mat_to_list` qui prend en argument une matrice d'adjacence `M` et renvoie la liste d'adjacence du même graphe.
Réponse : voir figure 24 en page 9.

Exercice 6 : Écrire une fonction `list_to_dict` qui prend en argument une liste d'adjacence `L` et renvoie le dictionnaire d'adjacence du même graphe.
Réponse : voir figure 25 en page 9.

Exercice 7 : Écrire une fonction `dict_to_mat` qui prend en argument un dictionnaire d'adjacence `D` et renvoie la matrice d'adjacence du même graphe.
Réponse : voir figure 26 en page 9.

```python
M1 = [[-1, 1, 1, -1, 1], [1, -1, 1, -1, 1], [1, 1, -1, -1, -1], [-1, -1, -1, -1, -1], [1, 1, -1, -1, -1]]
L1 = [[1, 2, 4], [0, 2, 4], [0, 1], [], [0, 1]]

def mat_to_list(M):
    n = len(M)
    L = []
    for i in range(n):
        L2 = [] # sous-liste des voisins de i
        for j in range(n):
            if M[i][j]!=-1:
                L2.append(j)
        L.append(L2)
    return L
print(mat_to_list(M1))
print(mat_to_list(M1)==L1)
```

FIGURE 24

```python
L1 = [[1, 2, 4], [0, 2, 4], [0, 1], [], [0, 1]]
D1 = {0: [1, 2, 4], 1: [0, 2, 4], 2: [0, 1], 3: [], 4: [0, 1]}
def list_to_dict(L):
    n = len(L)
    D = {}
    for i in range(n):
        D[i] = L[i]
    return D

print(list_to_dict(L1)==D1)
```

FIGURE 25

```python
M1 = [[-1, 1, 1, -1, 1], [1, -1, 1, -1, 1], [1, 1, -1, -1, -1], [-1, -1, -1, -1, -1], [1, 1, -1, -1, -1]]
D1 = {0: [1, 2, 4], 1: [0, 2, 4], 2: [0, 1], 3: [], 4: [0, 1]}
def dict_to_mat(D):
    n = len(D)
    M = [[-1 for j in range(n)] for i in range(n)]
    for i in range(n):
        for j in D[i]:
            M[i][j] = 1
    return M
print(dict_to_mat(D1))
print(dict_to_mat(D1)==M1)
```

FIGURE 26