Exercice 1.

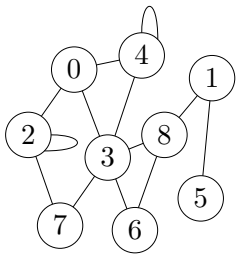


FIGURE 1

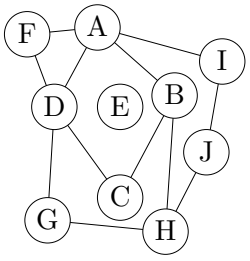


FIGURE 2

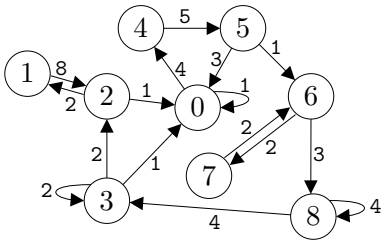


FIGURE 3

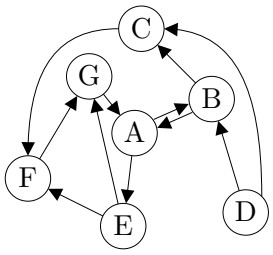


FIGURE 4

Matrice d'adjacence d'un graphe non orienté (partie facultative). Soit $n \in \mathbb{N}$ et $G = (S, A)$ un graphe non orienté avec $S = [\![0; n-1]\!]$. On suppose que G est donné par sa matrice d'adjacence « `M: list[list[int]]` ».

1. En Python, définir la matrice M pour le graphe de la figure 1.
2. Écrire une fonction qui prend en entrée `M` et renvoie une liste « `D: list[int]` » de taille n telle que `D[i]` est le degré du sommet `i`. On utilise ici la convention qu'une boucle dans G ajoute deux au degré du noeud associé, par exemple le sommet 2 de la figure 1 est de degré 4.
3. Écrire une fonction qui prend en entrée `M` et renvoie un booléen qui indique si la matrice `M` est symétrique. Votre fonction doit faire un minimum de tours de boucle.
4. Écrire une fonction qui prend en entrée un entier $n \in \mathbb{N}$ et renvoie la matrice d'adjacence associée au graphe complet à n sommets. Testez les fonctions des questions 2 et 3 sur les graphes obtenus.
5. Écrire une fonction qui prend en entrée `M` et renvoie un dictionnaire contenant les listes d'adjacence de G.
6. Précisez les complexités des fonctions précédentes.

Listes d'adjacence d'un graphe non orienté (partie obligatoire). Soit $G = (S, A)$ un graphe non orienté. On note `t` le type des éléments de S et on suppose que G est donné par un dictionnaire « `d: dict[t: list[t]]` » contenant les listes d'adjacence.

7. En Python, définir le dictionnaire `d` pour le graphe de la figure 2.
8. (a) Écrire une fonction qui prend en entrée `d` ainsi que deux sommets $(s_1, s_2) \in S^2$ et renvoie un booléen qui indique si s_1 et s_2 sont voisins dans G.
   (b) Écrire une fonction qui prend en entrée `d` et renvoie un booléen qui indique si G contient une boucle.
   (c) Précisez les complexités des deux fonctions précédentes.
9. (a) Écrire une fonction qui prend en entrée `d` et renvoie une liste « `A: list[(t,t)]` » contenant toutes les arêtes de G sans doublon (si `(s1,s2)` appartient à `A` alors `(s2,s1)` n'appartient pas à `A`). La fonction devra s'exécuter en temps $\Theta(\max(n, m))$ où n est le nombre de sommets et m le nombre d'arêtes.
   (b) Quel est le nombre d'arêtes dans le graphe complet à n sommets ? Vérifiez numériquement votre réponse en utilisant les fonctions précédentes.
10. (a) Dans cette question, on suppose disposer d'une liste « `L: list[list[t]]` » contenant toutes les listes d'adjacence de G. Cette liste `L` peut donc être définie par :

```
L = [d[s] for s in d]
```

    Écrire une fonction qui prend en entrée `L` et renvoie la liste de tous les sommets présents dans `L` (sans doublon). La complexité de votre fonction devra être en $\Theta(\max(n, m))$ où n est le nombre de sommets et m le nombre d'arêtes.
    (b) Expliquer pourquoi la fonction précédente ne renvoie pas nécessairement tous les sommets de S.

Soit $n \in \mathbb{N}$ le nombre de sommets dans G. On souhaite construire la matrice d'adjacence de G à partir de `d`. Une des difficultés est que l'ensemble des sommets n'est pas de la forme $[\![0, n-1]\!]$. Ainsi, on va commencer par associé à chaque sommet de G un identifiant $i \in [\![0, n-1]\!]$. La matrice d'adjacence $M = (m_{i,j})_{(i,j)\in[\![0,n-1]\!]^2}$ sera alors telle que $m_{i,j}$ vaut 1 si et seulement les sommets d'identifiants i et j sont voisins dans G. Pour pouvoir faire la traduction entre un identifiant et un sommet, on créé en plus un dictionnaire « `Id: dict[t: int]` » dont les clés sont les sommets de G et dont les valeurs sont les identifiants, ainsi qu'une liste « `S: list[t]` » telle que `S[i]` est le sommet d'identifiant `i`.

11. **(Facultatif)** Écrire une fonction qui prend en entrée `d` et renvoie M, `Id` et `S`. Quelle est la complexité de votre fonction ?

Matrice d'adjacence d'un graphe orienté pondéré (partie facultative). Soit G un graphe orienté pondéré avec n sommets. Formellement, G est donné par un triplet $G = (S, A, \omega)$ où $S = [\![0, n-1]\!]$ est l'ensemble des sommets, A est l'ensemble des arcs et $\omega : A \to \mathbb{R}_+^*$ est la fonction qui associe à chaque arc son poids. On suppose que G est donnée par sa matrice d'adjacence $M = (m_{i,j})_{(i,j)\in[\![0,n-1]\!]^2}$, c'est à dire que pour tout $a \in S^2$: $m_{i,j} = 0$ si $a \notin A$ et $m_{i,j} = \omega(a)$ si $a \in A$.

12. En Python, définir la matrice M pour le graphe de la figure 3.
13. Écrire une fonction qui prend en entrée la matrice M ainsi qu'une liste de sommets « `C: list[int]` » et renvoie :
    → Le poids de `C` si `C` est un chemin de G (le poids d'un chemin est la somme des poids de ses arêtes).
    → `None` si `C` n'est pas un chemin de G.

On souhaite maintenant construire les listes de successeurs pour G ainsi qu'un dictionnaire `omega` pour représenter la fonction de pondération ω. Le dictionnaire `omega` sera de type `dict[(int,int): float]`, ses clés seront tous les arcs $a \in A$ et les valeurs seront les $\omega(a)$.

14. Écrire une fonction qui prend en entrée la matrice M et renvoie une liste « `S: list[list[int]]` » contenant toutes les listes de successeurs ainsi que le dictionnaire « `omega: dict[(int,int): float]` ».
15. Précisez les complexités des fonctions précédentes.

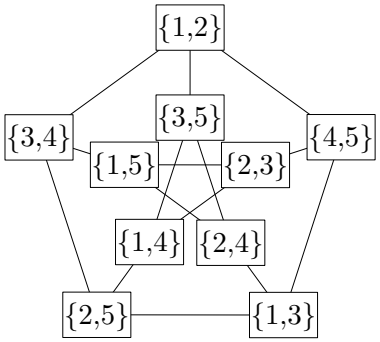
Listes de successeurs et de prédécesseurs d'un graphe orienté (partie obligatoire). Soit $G = (S, A)$ un graphe orienté et `t` le type des éléments de S. On suppose disposer d'un dictionnaire « `S: dict[t: list[t]]` » contenant les listes de successeurs de G.

16. En Python, définir le dictionnaire `S` pour le graphe de la figure 4.
17. Écrire une fonction qui prend en entrée `S` et renvoie un dictionnaire « `P: dict[t: list[t]]` » contenant les listes de prédécesseurs associées à G.
18. Précisez la complexité de la fonction précédente.

Exercice 2. Graphes de Kneser

Étant donnés deux entiers n, k tels que $0 \leqslant k \leqslant n$, on note $\mathcal{P}_k(n)$ l'ensemble des parties de $[\![1, n]\!]$ à k éléments. Le graphe de Kneser $KG_{n,k} = (S, A)$ est un graphe non orienté tel que $S = \mathcal{P}_k(n)$ et pour tout $(B_1, B_2) \in S^2$, B_1 et B_2 sont voisins dans $KG_{n,k}$ si $B_1 \cap B_2 = \varnothing$. Par exemple, avec $n = 5$ et $k = 2$, on obtient le graphe ci-contre.

Écrire une fonction qui prend en entrée les deux entiers n, k et renvoie le graphe $KG_{n,k}$. Le graphe sera représenté au choix par matrice d'adjacence ou par listes d'adjacence.



Exercice 3. Couverture par sommets

Soit $G = (S, A)$ un graphe non orienté et `t` le type des éléments de S. En Python, G sera représenté par un dictionnaire « `d: dict[t: list[t]]` » contenant les listes d'adjacence. On dira qu'un sommet $s \in S$ ***couvre*** une arête $a \in A$ si s est l'une des extrémités de a. Le problème de la couverture par sommets consiste à déterminer un ensemble $S' \subset S$ de cardinal minimal tel que toutes les arêtes de A soient couvertes par au moins un des sommets de S'.

Dans un premier temps, on propose d'utiliser un algorithme glouton pour résoudre le problème : initialement S' est vide, puis à chaque étape, on y ajoute le sommet de $S \setminus S'$ qui couvre le plus d'arêtes qui ne sont pas encore couvertes.

1. (a) Écrire une fonction qui prend en entrée `d` et renvoie une liste de type `list[t]` représentant l'ensemble S' obtenu avec l'algorithme glouton.
   (b) Montrer que la solution renvoyée par l'algorithme glouton n'est pas nécessairement optimale.

Afin de déterminer une solution optimale au problème de la couverture exacte, on va explorer toutes les solutions possibles et choisir celle de cardinal minimal. Pour cela, on part de $S' = \varnothing$ et on désigne par s l'un des sommets de G de degré non nul. Il y a alors deux possibilités à explorer :

→ On n'ajoute pas s à S'. On pose alors $G' = G$.

→ On ajoute s à S'. On définit alors G' le graphe obtenu à partir de G en y supprimant toutes les arêtes dont s est une extrémité. Formellement, $G' = (S, A')$ où A' est l'ensemble des arêtes $a \in A$ telles que $s \notin a$.

Dans les deux cas, il s'agit ensuite de trouver récursivement une couverture par sommets de G' sachant que s ne peut plus être sélectionné.

2. À l'aide de la méthode décrite ci-dessus, écrire une fonction qui renvoie une solution optimale au problème de la couverture par sommets.

Remarque. Le problème de la couverture par sommets est un problème NP-complet. Le fait de trouver un algorithme de résolution efficace est donc une question ouverte.