ISM MPSI/PCSI, Informatique
Année 2025/2026

TP Informatique 31

On complètera le fichier `TP31_GRAPHES2_std.py` en ligne sur le site de la classe.

Exercice 1

Le module `deque` de la bibliothèque `collections` permet de manipuler des objets de type `deque` conformes au modèle de file FIFO (First In First Out). On importe ce module avec `from collections import deque`. L'instruction `deque()` crée une `deque` vide et on ajoute ou supprime à droite et respectivement à gauche avec les méthodes `append`, `pop`, `appendleft` et `popleft`.

1. Construire `deque([1,2,3,4])` depuis une `deque` vide en alternant `append` et `appendleft`. On stockera le résultat dans une variable `a`.
2. Prédire puis vérifier le résultats de chacune des instructions exécutées consécutivement : `a.append(a.popleft())` et `a.appendleft(a.pop())`.

Exercice 2

Le *parcours en largeur* (BFS en anglais, Breadth First Search) s'applique indifféremment aux graphes orientés ou non orientés. Son principe est le suivant :

— on part d'un sommet qu'on place dans une file et que l'on considère comme « en cours de visite » ;
— tant que la file n'est pas vide, on sort le sommet premier entré, on fait entrer dans la file tous ses sommets adjacents « non visités » puis on considère le sommet d'origine comme « visité ».

Les états d'un sommet seront décrits par les chaînes de caractères :

— `"blanc"` pour non visité ;
— `"gris"` pour en cours de visite ;
— `"noir"` pour visité.

On utilisera les dictionnaires suivants :

— `couleur` : pour `s` un sommet, `couleur[s]` est sa couleur ;
— `dist` : pour `s` un sommet, `dist[s]` est la distance au sommet de départ ;
— `predec` : pour `s` un sommet, `predec[s]` est son prédécesseur dans le parcours en profondeur.

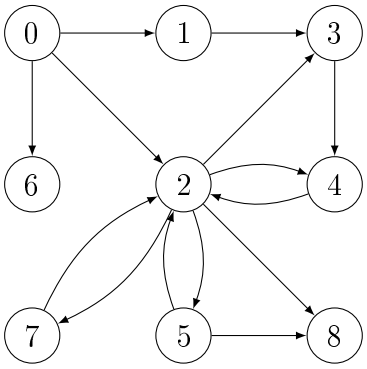
Pour manipuler une structure de file, on utilisera le module `deque` avec l'importation `from collections import deque`.

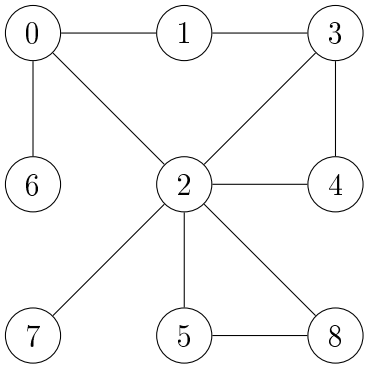
1. Écrire une fonction `bfs(S,A,s0)` d'arguments `S` la liste des sommets, `A` la liste des arêtes, `s0` un sommet et qui renvoie les dictionnaires `dist`, `predec`, `couleur` à l'issue du parcours en largeur du graphe $G = (S, A)$ depuis le sommet s_0.
2. Tester la fonction `bfs` sur le graphe orienté de l'exercice qui suit depuis le sommet 4.

Exercice 3

Soit $G = (S, A)$ un graphe orienté ou non. Le diamètre d'un graphe est la plus grande distance entre deux sommets du graphe, la distance étant la longueur du plus court chemin (en nombre d'arcs ou d'arêtes) entre les deux sommets.

1. Écrire une fonction `diametre(S,A)` d'arguments `S`, la liste des sommets, `A` la liste des arêtes et qui renvoie le diamètre du graphe $G = (S, A)$ et la liste de tous les couples de sommets distants du diamètre. Cette fonction utilisera la fonction de parcours en largeur `bfs`.
2. Tester la fonction `diametre` sur le graphes fournis en exemple ci-dessous.





Exercice 4

Un graphe $G = (S, A)$ orienté ou non est dit *biparti* s'il existe une partition $S = S_0 \sqcup S_1$ telle que toute arête (ou arc) du graphe a une extrémité dans S_0 et l'autre dans S_1.

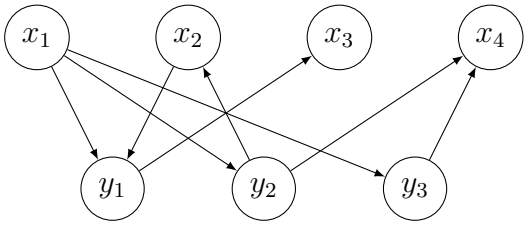


FIGURE 1 – Un graphe biparti

On peut montrer qu'un graphe est biparti si et seulement si on peut colorier ses sommets avec seulement deux couleurs sans que deux sommets adjacents aient la même couleur.

On se propose de tester le caractère biparti d'un graphe à l'aide d'un parcours en largeur. La couleur des sommets est initialisée à `None` dans le dictionnaire `color`. Puis, pour chaque sommet de couleur `None`, on bascule celle-ci à `True` et on lance un parcours en largeur. Lors de la visite d'un sommet auquel on accède depuis un autre, s'il est de couleur `None`, on le colorie dans la couleur opposée à celle du sommet d'origine et s'il est déjà coloré, on s'assure que sa couleur est conforme à l'alternance `True/False` avec son sommet d'origine.

1. Écrire une fonction `bipart_ds(A,s0,color)` d'arguments `A` le dictionnaire des arêtes, `s0` un sommet et `color` le dictionnaire des couleurs qui réalise le parcours en largeur du graphe depuis le sommet `s0`. La fonction renvoie `True` si elle parvient à réaliser un coloriage biparti depuis le sommet `s0` et `False` sinon.
2. Écrire une fonction `bipart_init_dfs(S,A)` d'arguments `S` la liste des sommets, `A` le dictionnaire des arêtes qui effectue le parcours en largeur du graphe $G = (S, A)$ et renvoie `True` s'il est biparti, `False` sinon. Tester la fonction sur les graphes fournis en exemple.