Graphes (Graphs)
Implémentations et parcours

1 Représentations / Implémentations

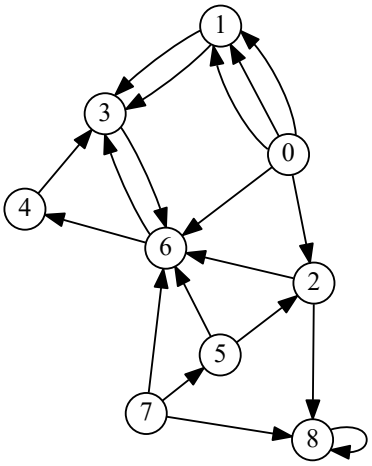

FIGURE 1 – Graphe orienté (Digraph) G'_1

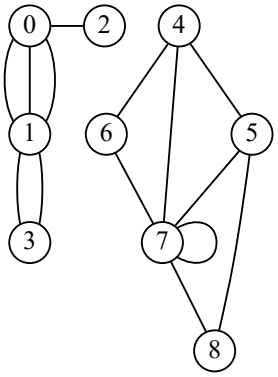

FIGURE 2 – Graphe non orienté (Graph) G'_2

Exercice 1.1 (`GraphMat` : Matrice d'adjacence)

On utilise pour cette première implémentation, la représentation par matrice d'adjacence.

1. Quelles sont les différences, pour l'implémentation, lorsque le graphe est orienté ou non orienté, valué ou non, possède des liaisons simples ou multiples ?

2. Donner les matrices d'adjacence représentant les graphes des figures 1 et 2.

3. Sachant que l'on veut pouvoir utiliser le même type pour représenter un graphe qu'il soit orienté ou non, simple ou 1-graphe, ou encore un multigraphe ou p-graphe, que doit contenir l'implémentation ?

Exercice 1.2 (`Graph` : Listes d'adajcence)

1. Quelle est l'autre manière de représenter / implémenter un graphe ?

2. Quelles sont les différences pour l'implémentation lorsque le graphe est orienté ou non orienté, possède des liaisons multiples ?

3. Donner les représentations des graphes des figures 1 et 2.

4. Sachant que l'on veut pouvoir utiliser le même type pour représenter n'importe quel type de graphe : orienté ou non, simple (ou 1-graphe) ou multigraphe (ou p-graphe) ; que doit contenir l'implémentation ?

Exercice 1.3 (Load)

Notre format de fichier `GRA` est un fichier texte contenant :
- une première ligne contenant 0 ou 1 : 0 pour non orienté, 1 pour orienté
- une seconde ligne contenant l'ordre du graphe
- une suite de lignes contenant les liaisons : deux numéros de sommets séparés par un espace

Voir les fichiers fournis : `digraph1.gra graph2.gra`.

Écrire les fonctions qui construisent un graphe à partir d'un fichier `".gra"` dans les deux implémentations.

Exercice 1.4 (Degrés)

1. Écrire une fonction qui construit deux vecteurs (listes en Python) *in* et *out* qui contiendront respectivement les demi-degrés intérieurs et extérieurs de tous les sommets d'un graphe orienté représenté par listes d'adjacence.

2. **Le degré** d'un graphe est le degré maximum de ses sommets.
   Le demi-degré extérieur d'un graphe est le demi-degré extérieur maximum de ses sommets.
   Le demi-degré intérieur d'un graphe est le demi-degré intérieur maximum de ses sommets.

   Écrire une fonction qui calcule les demi-degrés intérieur et extérieur d'un graphe orienté représenté par une matrice d'adjacence.

Exercice 1.5 (`dot`)

Écrire les fonctions qui construisent la représentation au format `dot` (simplifié) dans les deux implémentations.

Exemples :

- Graphe G'_1 (figure 1)

```
>>> print(todot(G1))
digraph {
  0 -> 1
  0 -> 2
  0 -> 6
  1 -> 3
  2 -> 6
  2 -> 8
  3 -> 6
  4 -> 3
  5 -> 2
  5 -> 6
  6 -> 3
  6 -> 4
  7 -> 5
  7 -> 6
  7 -> 8
  8 -> 8
}
```

- Graphe G'_2 (figure 2)

```
>>> print(todot(G2))
graph {
  1 -- 0
  1 -- 0
  1 -- 0
  2 -- 0
  3 -- 1
  3 -- 1
  5 -- 4
  6 -- 4
  7 -- 4
  7 -- 5
  7 -- 6
  7 -- 7
  8 -- 5
  8 -- 7
}
```

Bonus :
Écrire les fonctions qui construisent un graphe à partir d'un fichier `".dot"` (simplifié).

Notes : Sauf indications particulières, les graphes utilisés dans les exercices suivants seront des graphes simples pour les non orientés, et des 1-graphes sans boucles pour les orientés. Les exemples utilisés ici seront les graphes G_1 (1-graphe sans boucles issu de G'_1) et G_2 (graphe partiel simple issu de G'_2).

---

2 Parcours

Exercice 2.1 (Parcours en largeur)

1. Donner les forêts couvrantes obtenues lors des parcours largeur complets des graphes G_1 et G_2 à partir du sommet 0 puis à partir du sommet 7 (les sommets sont choisis en ordre croissant).
2. Donner le principe de l'algorithme de parcours largeur. Comparer avec le parcours d'un arbre général.
3. Comment stocker la forêt couvrante de manière linéaire ?
4. Écrire les fonctions de parcours largeur, avec construction de la forêt couvrante, pour les deux implémentations.

Exercice 2.2 (Parcours en profondeur)

1. Donner les forêts couvrantes obtenues lors des parcours profondeur des graphes G_1 et G_2 à partir du sommet 0 puis du sommet 7 (les sommets sont choisis en ordre croissant).
2. Donner le principe de l'algorithme récursif du parcours profondeur. Comparer avec le parcours d'un arbre général.
3. **Graphes non orientés**
   (a) Quels sont les différents types d'arcs rencontrés lors du parcours profondeur d'un graphe non orienté ?
   Classer les arcs du parcours profondeur de G_2 effectué en question 1 et ajouter les arcs manquants à la forêt.
   (b) Que faut-il ajouter au parcours profondeur pour repérer les différents types d'arcs ?
   (c) Écrire la fonction du parcours profondeur d'un graphe non orienté et représenté par une matrice d'adjacence. Indiquer au cours du parcours, le type des arcs rencontrés.
4. **Graphes orientés**
   (a) Quels sont les différents types d'arcs rencontrés lors d'un parcours profondeur ?
   Classer les arcs du parcours profondeur de G_1 effectué en question 1 et ajouter à la forêt couvrante les arcs manquants.
   Comment peut-on reconnaître le type d'un arc ?
   (b) On utilise la notion d'ordre préfixe de visite (première rencontre) et ordre suffixe de visite (dernière rencontre). En numérotant les sommets suivant ces deux ordres avec un unique compteur, écrire les conditions de classification des arcs.
   (c) Écrire la fonction du parcours profondeur d'un graphe orienté et représenté par listes d'adjacence. Indiquer au cours du parcours, le type des arcs rencontrés.
5. **Bonus**
   Le parcours profondeur peut être fait en itératif.
   Donner le principe d'un tel parcours et écrire la fonction correspondant pour un graphe orienté en représentation par listes d'adjacence.

3 Applications

Exercice 3.1 (Chemin – *Final S3# 2017*)

1. Comment trouver un chemin (ou une chaîne) entre deux sommets donnés dans un graphe ? Donner au moins deux méthodes différentes, les comparer.
2. Écrire une fonction qui cherche un chemin entre deux sommets. Si un chemin a été trouvé, il devra être retourné (une liste de sommets).

Exercice 3.2 (Distance au départ)

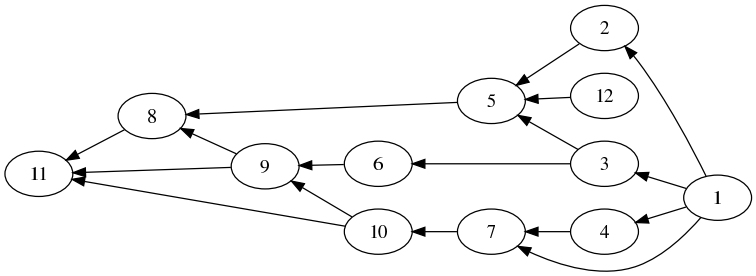

FIGURE 3 – Un graphe

L'objectif dans cet exercice est de connaître les sommets qui se trouvent, à partir d'un sommet de départ, à une distance dans un intervalle donné $[d_{min}, d_{max}]$. Le résultat sera affiché : si possible une ligne par niveau...

1. Calculer les distances depuis le sommet numéro 1 dans le graphe de la figure 3 (on veut toutes les distances des sommets atteignables, sans bornes).
2. Écrire la procédure `distances(`G`, `src`, `$dmin$`, `$dmax$`)` qui affiche les sommets à une distance comprise entre $dmin$ et $dmax$ du sommet src dans G graphe orienté (utiliser l'implémentation par matrice).

Exercice 3.3 (I want to be tree – *Final S3 2016*)

Définition :
Un **arbre** est un graphe **connexe sans cycle**.

Écrire la fonction `isTree` qui vérifie si un graphe non orienté est un arbre.

Exercice 3.4 (Diamètre – *Final S3 2016*)

Définitions :
- La **distance** entre deux sommets d'un graphe est le nombre d'arêtes d'une **plus courte chaîne** entre ces deux sommets.
- Le **diamètre** d'un graphe est la **plus grande distance** qu'il puisse exister entre deux de ses sommets.

Lorsque le graphe est un arbre, le calcul du diamètre est simple :
- À partir d'un sommet quelconque s_0, on cherche un sommet s_1 de distance maximale à s_0.
- À partir de s_1, on cherche un sommet s_2 de distance maximale à s_1.
- La distance entre s_1 et s_2 est le diamètre du graphe.

Écrire la fonction `diameter` qui calcule le diamètre d'un graphe qui est un arbre.

Et si le graphe n'est pas un arbre ?
— La méthode ci-dessus donne-t-elle le diamètre ?
— Si oui, justifier. Sinon que faudrait-il faire pour obtenir le diamètre ?

Exercice 3.5 (Compilation, cuisine. . .)

1. *Ordonnancement , un exemple simple :*
   Supposons l'ensemble d'instructions suivantes à effectuer par un seul processeur :

   ① lire(a)
   ② b ← a + d
   ③ c ← 2 * a
   ④ d ← e + 1
   ⑤ lire(e)
   ⑥ f ← h + c / e
   ⑦ g ← d * h
   ⑧ h ← e - 5
   ⑨ i ← h - f

   Quels sont les ordres possibles d'exécution ?
   Comment représenter ce problème sous forme de graphe ?

   Chaque solution correspond à un *tri topologique* du graphe.

2. Quelle doit être la propriété du graphe pour qu'une solution de tri topologique existe ?

3. Si on dessine le graphe en alignant les sommets dans l'ordre d'une solution de tri topologique, que peut-on constater ?

4. (a) Soit $suffix$ le tableau contenant les dates de dernière visite : l'ordre suffixe de tous les sommets de G lors d'un parcours en profondeur.
   Démontrer que pour une paire quelconque de sommets distincts $u, v \in S$, s'il existe un arc dans G de u à v, et si G a la propriété de la question 2, alors $suffix[v] < suffix[u]$.
   (b) En déduire un algorithme qui trouve une solution de tri topologique pour un graphe (on supposera qu'une solution existe).
   (c) Que faut-il changer à cet algorithme pour vérifier l'existence d'une solution dans un graphe quelconque ?
   (d) Écrire la fonction Python qui retourne une solution de tri topologique sous la forme d'une liste de sommets.

5. **Bonus :** Écrire une fonction qui vérifie si une liste de sommets représente une solution de tri topologique pour un graphe donné.

Et la cuisine dans tout ça ?