Les Graphes
Représentations et explorations

1 Représentations

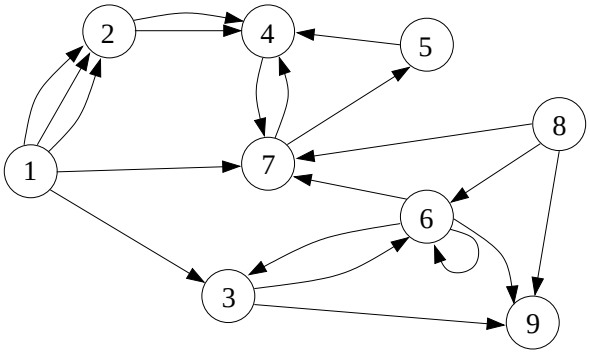

FIGURE 1 – Graphe G'_1

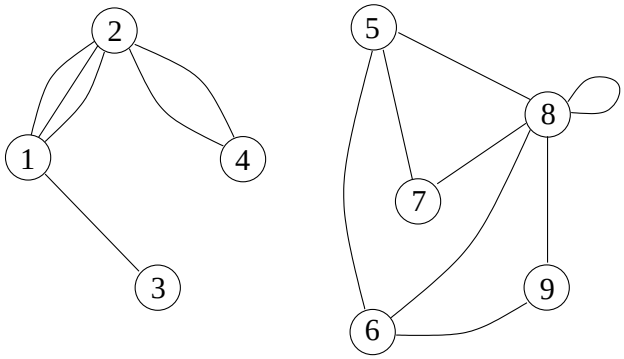

FIGURE 2 – Graphe G'_2

Exercice 1.1 (Représentation statique)

1. Comment s'appelle la représentation statique d'un graphe ?
2. Décrire cette représentation.
3. Donner les représentations statiques des graphes des figures 1 et 2.
4. Quelles sont les différences lorsque le graphe est orienté ou non orienté, valué ou non, possède des liaisons multiples ?
   Qu'est-ce que cela implique pour la représentation machine ?
5. Sachant que l'on veut pouvoir utiliser le même type pour représenter un graphe qu'il soit orienté ou non, simple ou 1-graphe et valué ou non, ou encore un multigraphe ou p-graphe (non valué dans ce cas), écrire le type de données statique `t_graphe_s` correspondant.

Exercice 1.2 (Représentation dynamique)

1. Quelle est l'autre manière de représenter un graphe ?
2. Décrire les différentes représentations selon que tel ou tel élément est statique ou dynamique.

   Nous choisissons ici d'avoir une représentation complètement dynamique. De plus, lors du parcours, nous voulons avoir un accès "immédiat" à l'extrémité d'un arc (ou d'une arête) emprunté !
3. Quelles sont les différences lorsque le graphe est orienté ou non orienté, valué ou non, possède des liaisons multiples ?
   Qu'est-ce que cela implique pour la représentation machine ?
4. Donner la représentation des sous-graphes issus de $S' = \{1, 2, 4, 5, 7\}$ des figures 1 et 2.
5. 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), valué ou non ; écrire le type de données dynamique `t_graphe_d` correspondant.
6. Écrire une fonction qui recherche le sommet n° s dans un graphe G représenté dynamiquement. On pourra supposer que la recherche est toujours positive ($1 \leq s \leq ordre(G)$).

Exercice 1.3 (Demi-degrés)

1. Rappeler les définitions du degré et des demi-degrés d'un sommet.
2. Calculer les degrés et les demi-degrés (quand ils existent) de tous les sommets des graphes des figures 1 et 2.
3. Écrire dans les deux représentations un algorithme qui remplit deux vecteurs *ddi* et *dde* qui contiendront respectivement les demi-degrés intérieurs et extérieurs de tous les sommets d'un graphe orienté.

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 boucle issu de G'_1) et G_2 (graphe partiel simple issu de G'_2).

---

2 Parcours

Exercice 2.1 (Parcours en largeur)

1. Donner, en précisant les forêts couvrantes obtenues, les parcours en largeur complets des graphes G_1 et G_2 à partir du sommet 8 (les sommets sont choisis en ordre croissant).
2. Donner le principe de l'algorithme de parcours en largeur. Comparer avec le parcours en largeur d'un arbre général.
3. Comment stocker la forêt couvrante de manière linéaire ?
4. Écrire les algorithmes de parcours en largeur, avec construction de la forêt couvrante, pour les deux représentations.

Exercice 2.2 (Parcours en profondeur)

1. Donner, en précisant les forêts couvrantes obtenues, les parcours en profondeur du graphe G_1 à partir du sommet 1, et du graphe G_2 à partir du sommet 5 (les sommets sont choisis en ordre croissant).
2. Donner le principe de l'algorithme récursif du parcours en profondeur. Comparer avec le parcours en profondeur d'un arbre général.
3. Quels sont les différents types d'arcs rencontrés lors d'un parcours en profondeur ?
   Classer les arcs des parcours en profondeur effectués en question 1 et ajouter aux forêts couvrantes les arcs manquants.
   Comment peut-on reconnaître le type d'un arc ?
4. On utilise la notion d'ordre préfixe de visite (première rencontre) et ordre suffixe de visite (dernière rencontre). En numérotant les sommets suivants ces deux ordres, écrire les conditions de classification des arcs.
5. Écrire l'algorithme récursif de parcours en profondeur, en indiquant au cours du parcours, quel est le type de l'arc rencontré, pour les cas suivants :
   (a) Le graphe est non orienté et représenté par une matrice d'adjacence.
   (b) Le graphe est orienté et représenté par listes d'adjacence.

(6.) Le parcours en profondeur peut être fait en itératif.
Donner le principe d'un tel parcours et écrire l'algorithme correspondant pour un graphe orienté en représentation dynamique.

3 Applications

Exercice 3.1 (Graphes bipartis – *Partiel décembre 2009*)

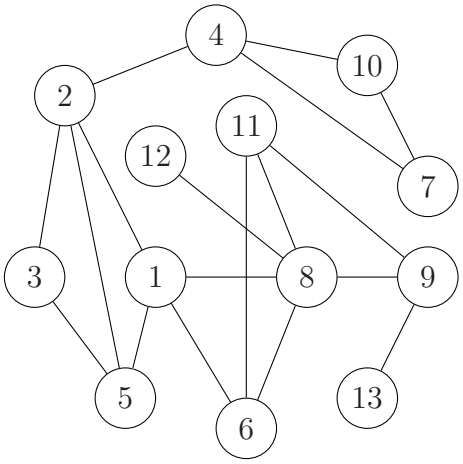

FIGURE 3 – Graphe G_3

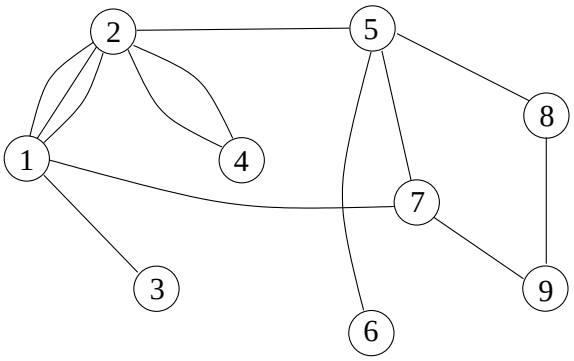

FIGURE 4 – Graphe G_4

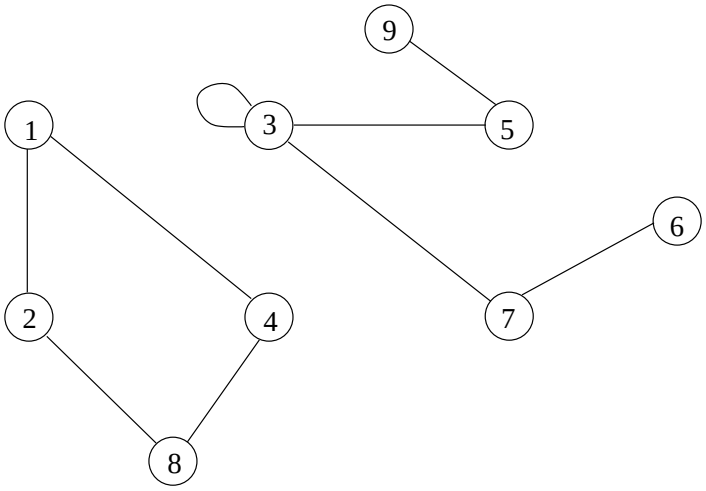

FIGURE 5 – Graphe G_5

Un graphe biparti est un graphe (un multigraphe) non orienté $G =< S, A >$, dans lequel S peut être partitionné en deux ensembles S_1 et S_2 tels que $(u, v) \in A$ implique soit que $u \in S_1$ et $v \in S_2$, soit que $u \in S_2$ et $v \in S_1$. Aucune arête ne doit relier deux sommets d'un même ensemble.

1. Les graphes des figures 3 à 5 sont-ils bipartis ? Pour chaque graphe biparti, donner les deux ensembles S_1 et S_2.

2. Écrire un algorithme qui teste si un graphe en représentation dynamique est biparti.

Exercice 3.2 (Poids cumulé d'un arbre couvrant – *partiel janvier 2011*)

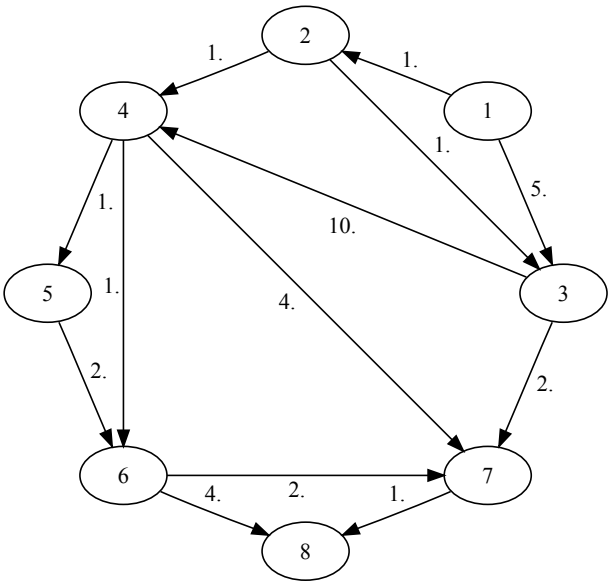

FIGURE 6 – Graphe orienté et valué

Dans cet exercice nous travaillerons avec des graphes **orientés** et **valués** en représentation dynamique.

On définit le poids cumulé d'un sommet dans un arbre couvrant (issu d'un parcours profondeur) comme la somme des poids des *fils* de sommet dans l'arbre plus le coût des arcs couvrant joignant ces sommets, c'est à dire : $p[s] = \sum_i(\texttt{cout}(s, s_i) + p[s_i])$ où s_i est un *fils* du sommet s dans l'arbre couvrant.

On se propose d'écrire un algorithme qui construit l'arbre couvrant du parcours **profondeur** d'un graphe et calcule le poids cumulé des sommets de cet arbre.

Par exemple, pour le graphe de la figure 6 en partant du sommet 2 (en rencontrant les sommets dans l'odre croissant) on obtiendra le vecteur de poids (présenté ici avec le vecteur de pères représentant l'arbre couvrant) suivant :

	1	2	3	4	5	6	7	8
pere	0	-1	2	3	4	5	6	7
poids	∞	17.	16.	6.	5.	3.	1.	0.

Remarque : les feuilles de l'arbre couvrant ont un poids de zéro et les sommets non atteints par le parcours ont un poids de $+\infty$.

Pour la suite, on considère que le principe d'un parcours profondeur est acquis (vous n'avez pas à décrire le principe du parcours, seulement ce qui est spécifique à l'algorithme.)

1. Écrire la fonction `cumul(ps,pere,poids)` qui effectue le parcours profondeur depuis le sommet pointé par `ps` et remplit le vecteur `pere` représentant l'arbre couvrant ainsi que le vecteur (réel) `poids` contenant les poids des sommets atteints par le parcours. La fonction renverra le poids cumulé du sommet `ps`.

2. Écrire la fonction (appel de l'algorithme précédant) `poids_cumul(s,g,pere,poids)` qui lance le parcours en profondeur sur le sommet `s` dans le graphe `g`.

Exercice 3.3 (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.

   Un autre exemple (Compilation) : Dans la plupart des langages de programmation, lorsqu'un programme se compose de plusieurs fichiers, certains fichiers, pour être compilés peuvent avoir besoin du résultat de la compilation d'autres fichiers. Cela impose de compiler les fichiers dans un ordre compatible avec leurs dépendances.

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

3. (a) Soit *os* 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, alors $os[v] < os[u]$.
   (b) En déduire un algorithme qui trouve une solution de tri topologique pour un graphe en représentation statique, 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 ?

Et la cuisine dans tout ça ?