Dans tout ce chapitre, on choisit de représenter les graphes par des tableaux de listes d'adjacence.

1 Parcours d'un graphe

1.1 Algorithme général

À partir d'un sommet donné dans un graphe, on peut construire un parcours injectif des sommets accessibles depuis ce sommet. L'idée générale est de tenir à jour deux ensembles : celui des sommets déjà rencontrés (ensemble fermé) et celui des sommets en attente (ensemble ouvert). L'ensemble fermé est celui des points qui ont été traités et sur lesquels on ne reviendra plus, l'ensemble ouvert est celui des sommets qu'on sait devoir traiter et mis en attente du traitement ultérieur.

- On initialise l'ensemble ouvert avec le sommet donné.
- Tant que l'ensemble ouvert n'est pas vide,
  - on retire un sommet de cet ensemble et on effectue le traitement voulu ;
  - on ajoute dans l'ensemble ouvert les sommets voisins du sommet en cours qui n'y sont pas déjà et qui ne sont pas dans l'ensemble fermé ;
  - on passe le sommet en cours dans l'ensemble fermé.

Il est clair qu'un tel algorithme termine, car le cardinal du complémentaire de l'ensemble fermé dans l'ensemble des sommets décroît strictement dans $\mathbb{N}$.

En fin d'algorithme, l'ensemble fermé contient tous les sommets accessibles depuis le sommet initial x (on les appelle aussi descendants de x), c'est-à-dire les sommets y tels qu'il existe un chemin de x à y.

Suivant la façon dont on retire un sommet de l'ensemble ouvert ou qu'on y ajoute les suivants, on obtient diverses formes de parcours. L'algorithme général ne présuppose rien sur le type abstrait ensemble.

Remarque. L'algorithme présenté ici fait partie de la catégorie des algorithmes dits « de marquage » : à chaque étape, les sommets ont un état (pas encore examine / ouvert / fermé). Pour suivre graphiquement le déroulement de l'algorithme, on marque les sommets d'une couleur selon leur état.

1.2 Complexité

On peut cependant donner une évaluation de la complexité d'un tel algorithme, si on fait quelques hypothèses raisonnables sur les opérations internes.

Par construction de l'algorithme, l'ensemble ouvert contient à chaque étape au maximum n éléments (les n sommets) et il en va de même pour l'ensemble fermé. Comme on retire un élément de l'ensemble ouvert à chaque étape, la boucle est donc parcourue au plus n fois. Hormis le coût du traitement du sommet $T(n)$, les autres opérations (ajouter un élément à un ensemble, vérifier si un élément appartient à deux ensembles de cardinal maximum n) peuvent être effectuées en complexité $O(n)$.

Donc la complexité de l'algorithme est du type $C(n) = nT(n) + O(n^2)$. En général, le coût du traitement est élémentaire ou linéaire, donc l'algorithme est de complexité quadratique.

Remarque. En fait, une implémentation plus fine de l'algorithme permet d'améliorer la complexité : si on suppose que le traitement est de coût constant, alors la complexité est en $O(n + m)$ où m est le nombre d'arêtes du graphe.

En effet, d'abord on choisit de représenter les graphes par des **listes d'adjacence**, pour avoir accès en temps constant aux liste de successeurs.

Ensuite, on peut exploiter le fait que les sommets sont connus et numérotés.

- Par exemple, on peut représenter l'ensemble fermé par un tableau de booléens : on marque la case i par "vrai" pour signifier que i appartient à cet ensemble, par faux sinon ; ajouter un élément ou vérifier qu'il est dedans se fait en complexité constante.
- Pour l'ensemble ouvert, c'est un peu plus compliqué : un couple (tableau de booléens, liste) peut faire l'affaire. Ajouter un objet à l'ensemble ouvert revient à ajouter un élément à la liste et à marquer une case du tableau par "vrai", retirer un objet revient à retirer la tête de la liste et à marquer une case du tableau par "faux", vérifier si l'ensemble est vide revient à vérifier si la liste est vide, vérifier qu'un objet est dans l'ensemble revient à lire une case du tableau. Tout ceci se fait en complexité constante.

Avec ces précautions, l'action "retirer un objet de l'ensemble ouvert et l'ajouter dans l'ensemble fermé" est effectuée au plus n fois, donc cette partie coûte $O(n)$.

Pour chaque sommet k, vérifier que ses successeurs sont ou non dans l'ensemble ouvert ou fermé a une complexité proportionnelle à s_k, nombre de successeurs de k (*i.e.* le degré sortant de k) : comme on effectue cette opération pour chaque sommet au plus une fois, on a une complexité en $O(\sum_{k \text{ sommet}} s_k) = O(m)$, nombre d'arcs du graphe.

Au total, on a une complexité en $O(n+m)$.

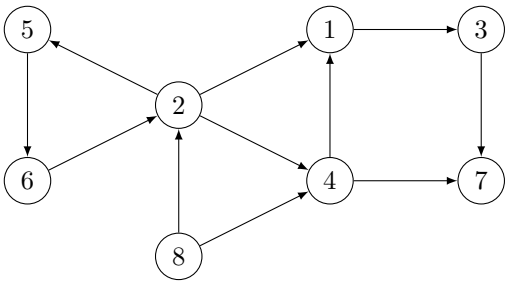
Une autre façon de faire est de marquer les sommets : on leur associe des états (ouvert, fermé, repos), qu'on stocke dans un tableau.

Mais comme $m = O(n^2)$, on retrouve le calcul grossier précédent.

1.3 Parcours en profondeur d'abord (DFS - Depth First Search)

Dans ce parcours, on choisit de représenter l'ensemble ouvert par une **pile**. Les sommets traités en premier sont donc les derniers entrés : de cette façon, on a tendance à explorer les sommets les plus éloignés du sommet initial avant ses plus proches voisins.

Donnons un exemple avec le graphe suivant, qu'on parcourt à partir du sommet 8 : *cf* document annexe



1.4 Parcours en largeur d'abord (BFS - Breadth First Search)

Dans ce parcours, on choisit de représenter l'ensemble ouvert par une **file**. Les sommets traités en premier sont donc les premiers entrés : de cette façon, on a tendance à explorer les sommets les moins éloignés du sommet initial avant ses plus lointains voisins.

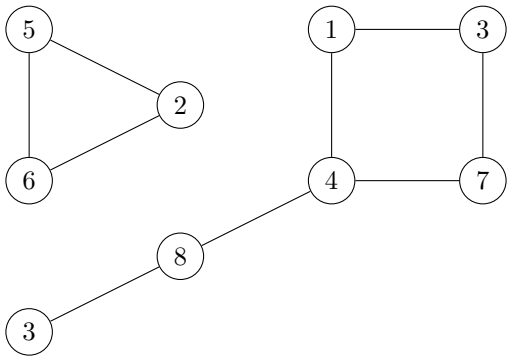
Donnons un exemple avec le graphe précédent, qu'on parcourt à partir du sommet 8 : *cf* document annexe

2 Connexité d'un graphe

2.1 Graphes connexes

Définition. Un graphe non orienté est connexe si et seulement si pour tout couple $(x, y) \in S^2$ tel que $x \neq y$, il existe une chaîne d'extrémités x et y.

Un graphe orienté est dit connexe quand le graphe non orienté sous-jacent est connexe.



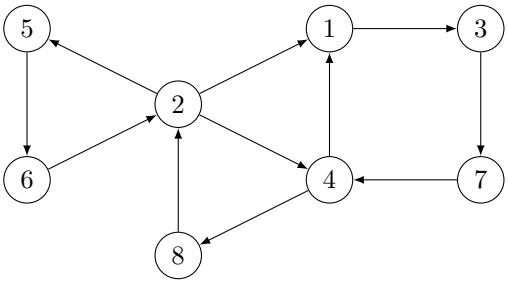
Graphe non connexe

La question de savoir si un graphe est connexe ou pas est une question importante : comment être sûr que votre machine (ou une quelconque autre) peut bien accéder à toutes les ressources du net ? On dispose d'une condition nécessaire (mais largement insuffisante).

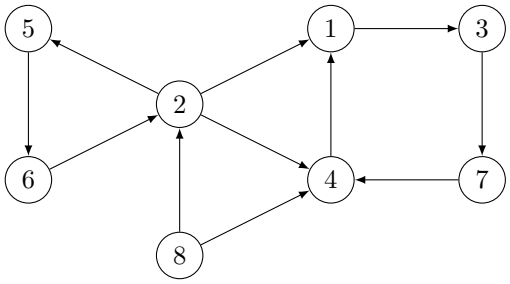
Proposition 1 *Si G est un graphe connexe d'ordre n, alors il a au moins $n-1$ arêtes.*

2.2 Graphes fortement connexes

Définition. Un graphe **orienté** est fortement connexe si et seulement si pour tout couple $(x, y) \in S^2$ tel que $x \neq y$, il existe un chemin d'origine x et de destination y.



graphe fortement connexe



graphe connexe, non fortement connexe

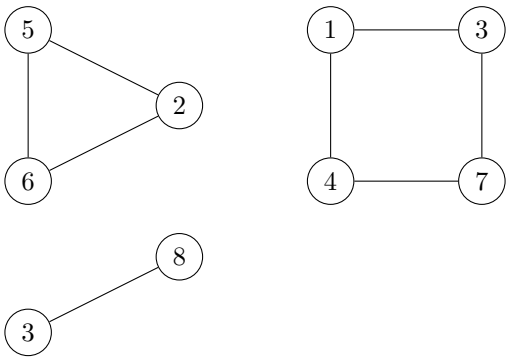
Il y bien sûr un lien évident entre les notions : tout graphe orienté fortement connexe est connexe, mais la réciproque est fausse.

2.3 Composantes connexes

Définition. Soit $G = (S, A)$ un graphe. Un sous-graphe de G est une graphe $G' = (S', A')$ tel que $S' \subset S$, $A' \subset A$ et pour tout $a \in A'$, les extrémités de a appartiennent à S'.

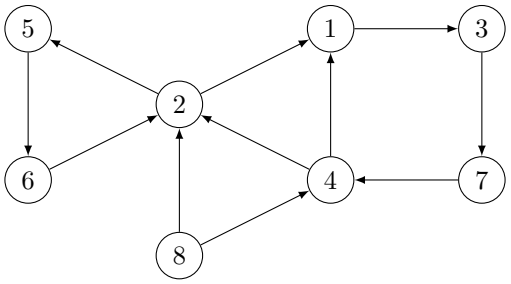
Autrement dit, on obtient un sous-graphe d'une graphe en supprimant des sommets et les arêtes qui ont pour extrémités les sommets supprimés.

Définition. Soit G un graphe. On appelle composante connexe du graphe tout sous-graphe de G, maximal pour la connexité, c'est-à-dire tout sous-graphe connexe de G qui n'est contenu dans aucun autre sous-graphe connexe de G.

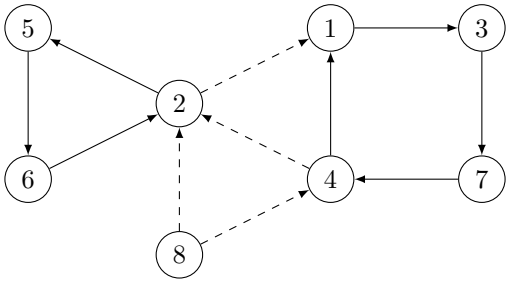


Graphe à trois composantes connexes

On a bien sûr la notion correspondante de composante fortement connexe d'un graphe orienté, c'est-à-dire tout sous-graphe orienté maximal pour la forte connexité.



Les trois composantes fortement connexes du graphe précédent



3 Calcul des composantes connexes

3.1 Question de la connexité d'un graphe

Si on effectue un parcours du graphe non orienté à partir d'un sommet, alors l'ensemble fermé contient tous les sommets accessibles à partir du sommet donné : on a donc déterminé les sommets qui appartiennent à la composante connexe du sommet.

Grâce à ces calculs, on peut alors répondre facilement à la question de la connexité d'un graphe : si l'ensemble fermé contient tous les sommets, alors le graphe est connexe. Comme on ne fait aucun traitement sur les sommets, on peut donc répondre à cette question en $O(n^2)$ opérations élémentaires (en fait $O(n+m)$).

3.2 Calcul des composantes connexes

En réutilisant les algorithmes de parcours de graphe, on peut construire un algorithme de calcul des composantes connexes d'un graphe non orienté.

- Tant que le graphe n'est pas vide,
  - on choisit un sommet ;
  - on parcourt le graphe à partir de ce sommet : on récupère l'ensemble des sommets appartenant à sa composante connexe, qu'on stocke dans un ensemble ;
  - on supprime les sommets ainsi calculés (et les arêtes associées).

Le cardinal du graphe diminuant strictement dans $\mathbb{N}$ à chaque itération, l'algorithme termine.

En fin d'algorithme, on obtient l'ensemble des ensembles de sommets d'une même composante connexe. En toute rigueur, il faudrait en fait reconstituer chaque composante connexe en récupérant les arêtes entre les sommets de chaque ensemble, ce qui peut être fait durant le parcours : quand on passe en revue les successeurs, on décrit en fait les arêtes depuis un sommet.

Là encore, la complexité est quadratique. Soit k le nombre de composantes connexes. Sur chaque composante connexe de cardinal c_i, le parcours du graphe va engendrer un ensemble en $O(c_i^2)$ opérations, donc le coût total du calcul des ensembles est $O(\sum_{i=1}^{k} c_i^2)$. Or on a évidemment $n = \sum_{i=1}^{k} c_i$ donc $n^2 = \sum_{i=1}^{k} c_i^2 + 2 \sum_{1 \leqslant i < j \leqslant k} c_i c_j \geqslant \sum_{i=1}^{k} c_i^2$, donc le coût du calcul des k ensembles est en $O(n^2)$. Comme le graphe comporte n sommets et au plus n^2 arêtes, supprimer les sommets et les arêtes se fait en $O(n^2)$ opérations, donc le coût total est en $O(n^2)$.

Remarque. Encore une fois, ce sont des estimations grossières de la complexité. Une analyse plus poussée donnerait encore $O(n+m)$ opérations élémentaires.

4 Implémentation

Les algorithmes précédents, exprimés en langage courant, cachent toutes les difficultés d'implémentation concrète.

D'abord, il est plus simple de représenter les graphes par des listes d'adjacence.

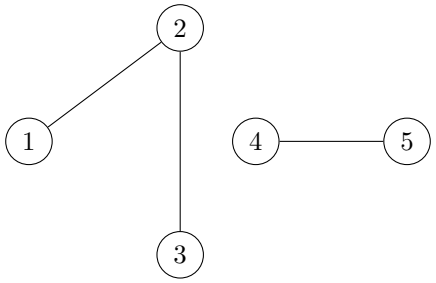
Ensuite, les ensembles ouverts, fermés peuvent être astucieusement représentés par des tableaux de booléens ou des couples (tableau de booléens, objet de type liste), ou être codés par des tableaux de marquage des sommets.

Enfin, il est hors de question pour calculer des sous-graphes de renuméroter les sommets dans chaque composante connexe, car il faudrait garder la trace des changements de numérotations pour s'y retrouver. Une solution est d'enrichir nos représentations à base de matrices ou de listes d'adjacence. L'ensemble des sommets est fixé au départ et les sous-graphes ont leurs sommets pris parmi ceux-là.

a) Matrice d'adjacence

On représente toujours un graphe par une matrice d'adjacence, mais on ajoute un tableau supplémentaire b de booléens, qui indique si le sommet numéroté i appartient au graphe ou pas.

Par exemple, avec le graphe suivant



le graphe complet a pour représentation le couple (b, m) où $b = [V, V, V, V, V]$ et $m = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$

ses deux composantes connexes sont représentées par exemple par les couples (b_1, m_1) et (b_2, m_2) :

$b_1 = [V, V, V, F, F]$ et $m_1 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$, $b_2 = [F, F, F, V, V]$ et $m_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$

Il faut donc prendre en compte une deuxième information avant d'interpréter le contenu d'une case $m[i, j]$ de la matrice : si $b[i] = F$, alors le contenu de $m[i, j]$ est non signifiant, sinon il s'interprète comme d'habitude.

On peut même aller jusqu'à fixer la matrice m et à manipuler les sous-graphes à partir des tableaux t seulement si on ne change pas les arêtes.

b) Listes d'adjacence

On fait de même : si t est le tableau des listes d'adjacence du graphe initial, avec un tableau de booléens b par sous-graphe, on interprète le contenu de la liste $t[i]$.

Si $b[i]$ et $b[j]$ sont vrais (ce qui signifie que i et j sont deux sommets du graphe), alors j appartient à $t[i]$ si et seulement si (i, j) est un arc du graphe.

Sinon, la présence ou non de j dans la liste $t[i]$ ne signifie rien, car l'un des deux sommets i ou j n'appartient pas au graphe.

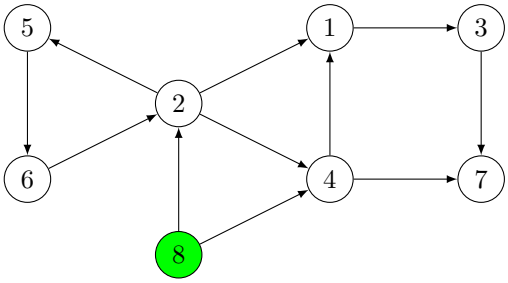
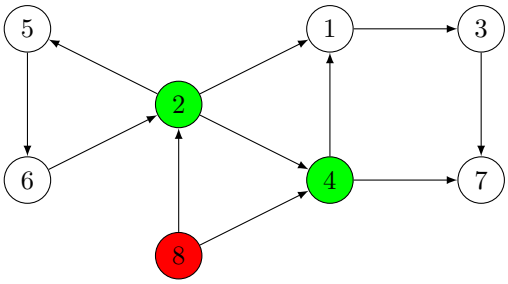
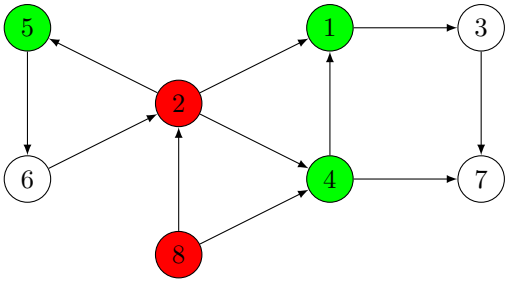
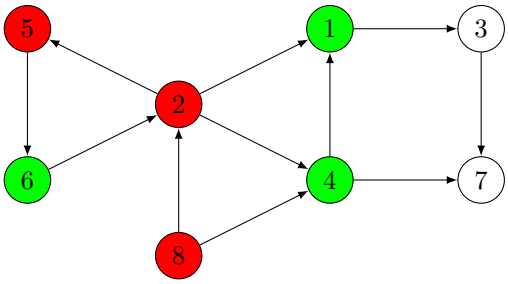
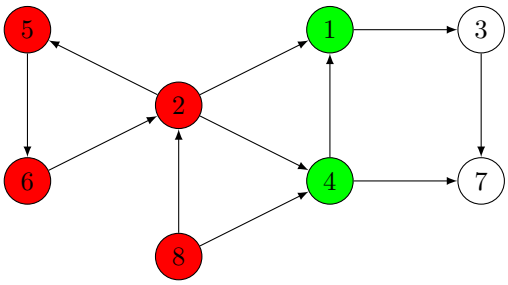
Le graphe précédent est représenté par le couple (b, t) où $b = [V, V, V, V, V]$ et $t = [[2], [1, 3], [2], [5], [4]]$.

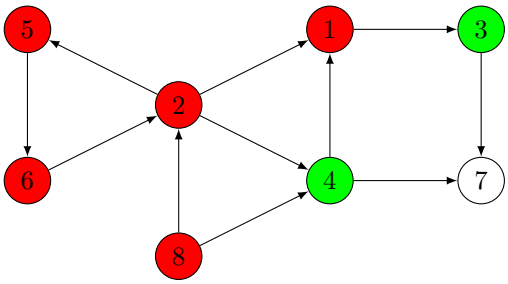
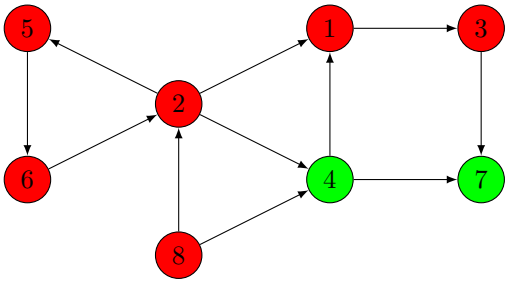
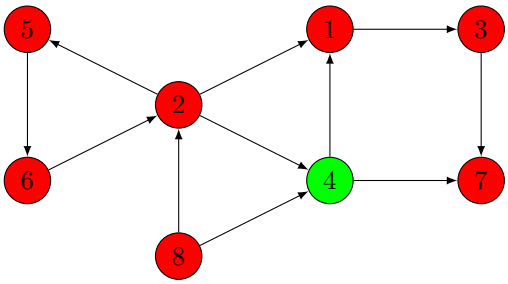
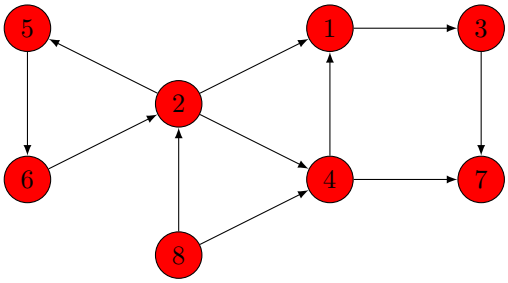
Ses deux composantes connexes sont représentées par les couples (b_1, t_1) et (b_2, t_2) :

$b_1 = [V, V, V, F, F]$ et $t_1 = [[2], [1, 3], [2], [\,], [\,]]$

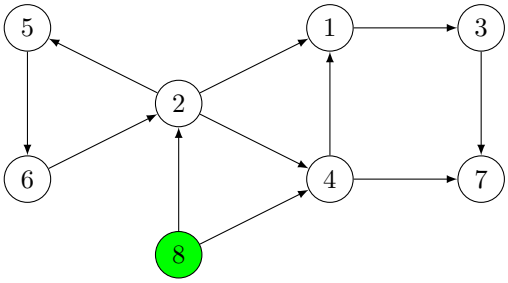
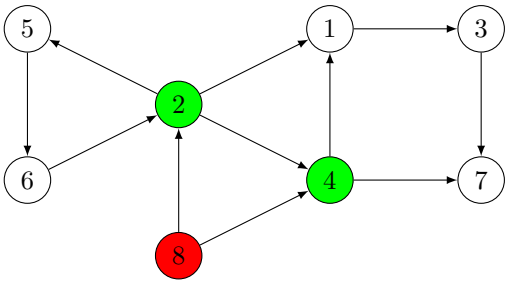
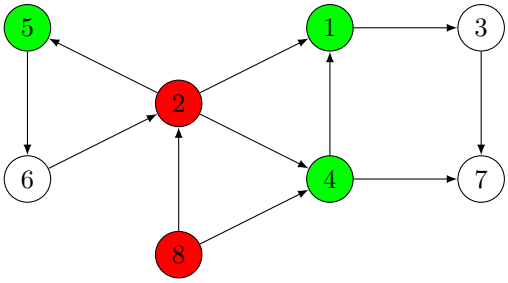
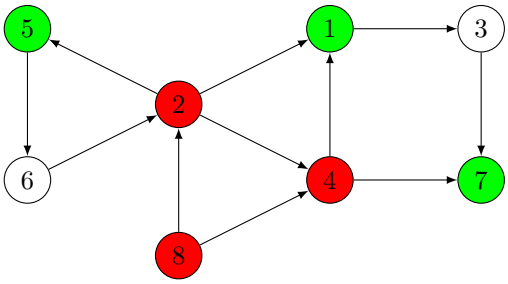
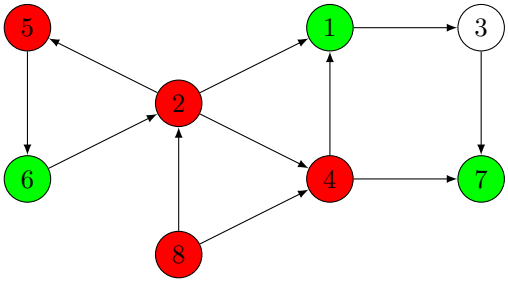
$b_2 = [F, F, F, V, V]$ et $t_2 = [[\,], [\,], [\,], [5], [4]]$

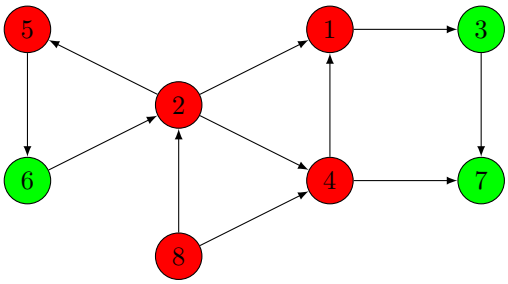
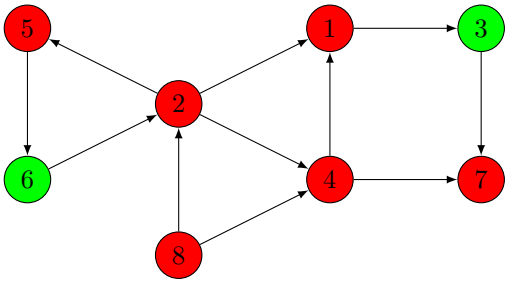
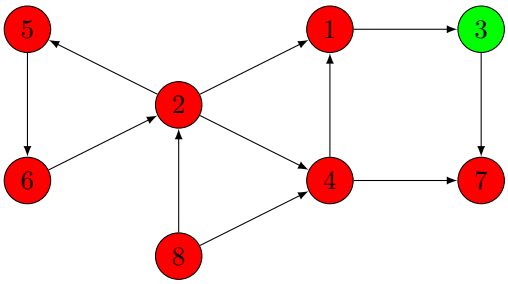
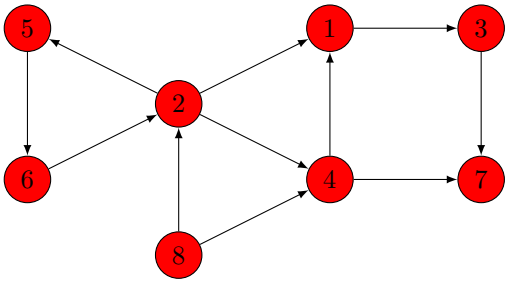
A Parcours DFS

graphe	ensemble ouvert	ensemble fermé
	8	
	2,4	8
	5,1,4	8,2
	6,1,4	8,2,5
	1,4	8,2,5,6

graphe	ensemble ouvert	ensemble fermé
	3,4	8,2,5,6,1
	7,4	8,2,5,6,1,3
	4	8,2,5,6,1,3,7
		8,2,5,6,1,3,7,4

B Parcours BFS

graphe	ensemble ouvert	ensemble fermé
	8	
	2,4	8
	4,5,1	8,2
	5,1,7	8,2,4
	1,7,6	8,2,4,5

graphe	ensemble ouvert	ensemble fermé
	7,6,3	8,2,4,5,1
	6,3	8,2,4,5,1,7
	3	8,2,4,5,1,7,6
		8,2,4,5,1,7,6,3