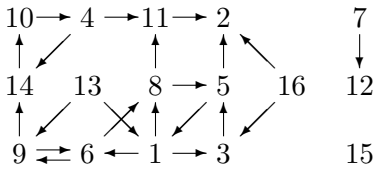
# Graphes, Feuille de TD $N^o$ 2 : Parcours

Rappel : parcours en profondeur de tout le graphe :

```
DFS(G) :
pour tous les sommets s, faire c[s] <- bleu
Initialisation()
pour tous les sommets s, faire
     si c[s] = bleu  alors  Depart()
                            Rec_DFS(s, ArgDuMain)
                            Arrivee()
Conclusion()

Rec_DFS (t, AutresArguments) :  /* t est suppose etre bleu */
c[t] <- blanc
Prefixe()
pour tous les successeurs/voisins u de t
       faire selon c[u] :  bleu :  PrefixeExt()    // variante de placement de Prefixe
                                   Rec_DFS(u, ArgDuRec)
                                   SuffixeExt()   // variante de placement de Suffixe
                           blanc : Blanc()
                           rouge : Rouge()
c[t] <- rouge
Suffixe()
```




1. Faire un DFS sur le graphe ci-contre. Donnez la forêt couvrante, le type des arcs et les numéros préfixe et suffixe des sommets. les sommets seront ordonnés par leur numéros.
2. Utilisez un DFS pour décider si un graphe non-orienté est biparti.
3. Donnez un algorithme linéaire qui détecte si un graphe $G$ orienté possède un cycle, et dans l'affirmative, qui en rend un.
4. Donnez un algorithme linéaire qui détecte si un graphe non-orienté est acyclique.
5. Donnez un algo linéaire qui prend en entrée un graphe $G$ orienté avec les sommets numérotés de 1 à n, et qui remplit un tableau `MaxAncetre[sommet] de sommet`, où, pour tout sommet $s$

   $MaxAncetre(s) = max\{t \mid$ il y a un chemin de $t$ vers $s\}$
6. Complétez le code du parcours en profondeur pour remplir le tableau $TypeDArc[arc]$ qui donne le type de tout arc parmi $ArcDeLaForet$, $ArcVersLaGauche$, $ArcAscendant$, $ArcDescendant$.
7. Donner le pseudo-code d'une fonction qui prend en entree un graphe orienté $G$, supposé être sans cycle, deux sommets $x$ et $y$ et rend le nombre de chemins distincts de $x$ à $y$.
8. Où se trouvent dans la forêt couvrante les premiers et les derniers sommets numérotés en préfixe et en suffixe ?
9. (maison) Comment sont répartis les sommets d'une même CFC dans la forêt couvrante d'un DFS ?

   On fait une numérotation postfixe des sommets lors d'un DFS. Soit $x$ le dernier sommet numéroté. Soit $y$ un sommet. Montrez qu'un sommet $y$ est dans la CFC de $x$ ssi il est co-accessible depuis $x$. Comment trouver les sommets de la CFC de $x$ ?

   Soit $G_x$ le sous-graphe obtenu en enlevant à $G$ les sommets de la CFC de $x$.
   Quel liens y-a-t-il entre les CFC de $G$ et celles de $G_x$ ?
   La forêt du DFS de $G$, privée des sommets de la CFC de $x$, est-elle la forêt d'un DFS de $G_x$ ?
   Soit $x'$ le dernier sommet numéroté qui n'est pas dans la CFC de $x$. Comment trouver les sommets de la CFC de $x'$ ?

   Donnez un algo linéaire pour trouver les CFC d'un graphe.
10. (maison) Donner un algorithme de tri topologique qui n'utilise pas de parcours. Analyser le coût de cet algorithme. Quelles structures de données peut-on utiliser pour faire le travail efficacement?

```
Boucle_BFS(s)
             Depart()
             c[s] <- blanc
             ajouter s a la file
             tant que la file n'est pas vide
                   extraire t de la file
                   c[t] <- rouge
                   Sortie()
                   pour tous les successeurs/voisins u de t faire
                          selon c[u] :
                                bleu  : c[u] <- blanc
                                        ajouter u a la file
                                        Entree()
                                blanc : Blanc()
                                rouge : Rouge()
             Arrivee()
```

parcours en largeur depuis $s_0$ :

```
BFS(G, s0) :
pour tous les sommets s, faire c[s] <- bleu
file <- vide
Initialisation()
Boucle_BFS(s0)
Conclusion()
```

parcours en largeur de tout le graphe :

```
BFS(G) :
pour tous les sommets s, faire c[s] <- bleu
file <- vide
Initialisation()
pour tous les sommets s, faire
      si c[s] = bleu
      alors   Boucle_BFS(s)
Conclusion()
```

11. Utilisez un BFS pour aire un calcul de plus court chemin (les arc étant tous de poids 1) depuis le sommet 13 sur le graphe de l'exo 1. Les sommets seront ordonnés par leur numéros.

12. Utilisez un BFS pour décider si un graphe non-orienté est biparti.

13. Donner un algorithme qui prend en entrée un graphe non-orienté $G$, un sommet $s_0$ et calcule $CardPCC[x]$ pour chaque sommet $x$, $CardPCC[x]$ étant le nombre de plus courts chemins (en nombre d'arêtes, le graphe n'est pas valué) de $s_0$ à $x$. (Exemple, si $G$ est un 3-cube, et si $s_0$ et $x$ sont diamétralement opposés, alors $CardPCC[x] = 6$)

14. Dans un graphe non-orienté, les sommets sont coloriés dans l'une des trois couleurs vert, gris, violet.

    Donner un algorithme linéaire qui trouve deux sommets, l'un vert, l'autre violet, les plus proches possibles. Les distances sont en nombre d'arêtes.

15. (maison) Plus courts chemins avec poids 0 ou 1 : On a un graphe orienté dont les arcs sont valués par des poids. Pour tout arc, le poids est 0 ou 1. Il y a un sommet source $s$. On veut determiner pour chaque sommet $x$ la distance de $s$ vers $x$ (valuation du meilleur chemin de $s$ à $x$).

    (a) Rappeler comment obtenir efficacement les distances quand toutes les valuations sont égales à 1.

    (b) Donner un algorithme permettant de trouver tous les sommets à distance 0.

    (c) Décrire comment trouver tous les sommets à distance 1.

    (d) Donner un algorithme efficace donnant les distances (expliquer puis donner un pseudo-code). Donner la complexité de votre algorithme.