M23 Graphes - épisode 2

1 - Composantes connexes

> ***Dans le fichier `algos.c`, écrire le code de la fonction suivante, qui retourne le nombre de composantes connexes du graphe reçu en argument :***
>
> ```
> size_t nb_composantes_connexes(graphe const *g);
> ```

Travail de groupe *: avant de coder, commencer par proposer un algorithme pour déterminer le nombre de composantes connexes d'un graphe.*

Quelques pistes :

- on aura besoin de matérialiser le fait qu'un sommet a déjà été visité ou non
- il faudra faire un parcours des sommets du graphe de proche en proche sur les sommets non-visités. Pour cela on pourra au choix :
  - écrire un algorithme récursif
  - maintenir un "front" de sommets actifs à partir desquels on peut encore continuer l'exploration du graphe

Pour stocker une information associée à chaque sommet d'un graphe, on peut déclarer un tableau de valeurs dont la taille correspond à l'ordre du graphe. Comme les sommets sont numérotés de `0` à `nb_sommets - 1`, la case d'index `i` contient les données associées au sommet `i`.

Par exemple, pour créer un tableau de `bool` de taille `N` avec la valeur `true` dans toutes les cases, on peut écrire le code suivant :

```
bool *tab = malloc(sizeof(bool) * N);
for (size_t i = 0; i < N; i++)
    tab[i] = true;
```

> ***! Ne pas oublier de systématiquement libérer la mémoire allouée dynamiquement !***

Pour gérer un "front" de sommets, on peut se servir d'un tableau dans lequel on va stocker les `sommet` actifs à partir desquels il faut continuer l'exploration du graphe.

---

2 - Sommets connectés

> ***Dans le fichier `algos.c`, écrire le code de la fonction suivante qui détermine si deux sommets d'un graphe sont connectés par une chaîne :***
>
> ```
> bool sont_connectes(graphe const *g, sommet s1, sommet s2);
> ```

Il faudra tout d'abord s'assurer que les 2 sommets `s1` et `s2` existent bien dans le graphe `g`.