Cette *vingt-et-unième* colle vous fera écrire des fonctions sur des graphes représentés par des listes d'adjacence en OCaml, sur les colorations comme étudiées en Ex.4 du DS 05 mercredi dernier.

On pourra travailler depuis Windows avec `https://BetterOCaml.ml/`.

Ex.1 Graphes représentés par listes d'adjacence - OCaml (10 minutes)

Comme à la colle 20 de la semaine dernière, on considère dans toute cette colle des graphes *orientés* $G = (S, A)$, représentés par listes d'adjacence.

1. Proposer un type (non récursif) en OCaml permettant de représenter un tel graphe (orienté) $G = (S, A)$ ayant $n = |S| \in \mathbb{N}$ sommets, numérotés dans l'ordre $S = [\![0; n-1]\!]$, et des arcs A.

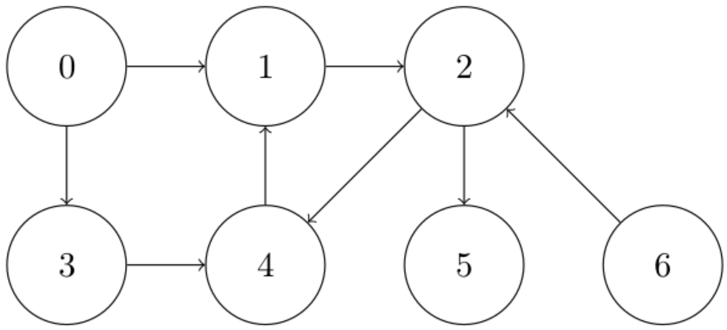

Figure 1 – Un petit graphe G_1.

2. Implémenter en OCaml le graphe G_1 représenté dans la figure ci-dessus comme une variable `g1`, sur laquelle vous pourrez tester les questions suivantes.
3. Écrire une fonction `nombre_sommets : graphe -> int` qui calcule le nombre de sommets d'un graphe G représenté comme expliqué précédemment.
4. Même question pour une fonction `successeurs : graphe -> int -> int array` qui renvoie le tableau des successeurs d'un sommet donné. Par exemple `successeurs g1 0` doit renvoyer le tableau `[| 1; 3 |]` (notez que leur ordre n'a pas d'importance).

Ex.2 Coloration de graphes représentés par listes d'adjacence - OCaml (40 minutes)

On considère la même représentation de graphes qu'à l'exercice d'avant.

5. Dessiner au brouillon le graphe G_1 de la Figure~1 et ajouter des couleurs à chaque sommet telles que la coloration proposée soit valide pour le graphe. Trouver une coloration ayant un nombre k minimal de couleurs. Que vaut k ?
6. On rappelle que, formellement, une k-coloration (valide ou non) d'un graphe $G = (S, A)$ est une fonction de S dans $[\![0, k-1]\!]$.
   — Définir en OCaml un type de données `type coloration = ...`, le plus simple possible, qui représente une coloration.

- Ce type ne cherche pas à être attaché à un graphe, en particulier on ne modifiera pas le type `graphe` !
- Il faut pouvoir lire et modifier en temps constant la couleur de n'importe quel sommet (connu par son numéro $s = 0, \dots, |S \vdash 1)$.

7. Représenter en OCaml une constante `coloration_g1` qui encode la k-coloration que vous avez proposé pour le graphe G_1 à la question 5. et selon cette représentation de la question 7. précédente.
8. Rappeler la définition de la validité d'une coloration (pour un graphe donné).
   - Implémenter cette définition en une fonction `est_valide (g:graphe) (c:coloration) : bool`.
   - Quelle est sa complexité temporelle en fonction de $n = |S|$ le nombre de sommets du graphe et/ou $m = |A|$ le nombre d'arcs du graphe ?
9. Implémenter une fonction `est_1_coloriable (g:graphe) : bool` qui décide, en temps au plus $\mathcal{O}(n + m)$, si le graphe `g` est 1-coloriable (i.e., coloriable par une coloration utilisant $k = 1$ couleur différente).
   - Tester la sur le graphe exemple `g1`. Est-il 1-coloriable ?
10. On cherche à écrire une fonction `toutes_les_colorations (n:int) (k:int) : coloration list` (récursive) qui génère toutes les colorations candidates d'un graphe G a $n = |S|$ sommets et k couleurs différentes. On pourra décomposer en trois cas :
    1. si $n = 0$, il faut renvoyer une liste vide (aucune coloration de 0 sommet),
    2. si $n = 1$, il faut renvoyer la liste des k colorations de 1 sommet mettant ce sommet à $c(1) = 0$, ou $c(1) = 1$, ou $\dots$, ou $c(1) = k - 1$ (vous pouvez utiliser `List.init k (fun i -> [i])`),
    3. et si $n > 1$, on peut commencer par calculer la liste des colorations des sommets de numéro 1 à $n - 1$, par un appel récursif à `let colorations_1_nmoins1 = toutes_les_colorations (n-1) k`, puis considérer une liste `res` initialement vide de colorations (une référence de type `(coloration list) ref`). Pour chaque couleur possible pour le sommet de numéro 0 (`couleur0 = 0 .. k-1`, dans une boucle `for`), il faut ajouter à `res` toutes les colorations de la forme `couleur0 :: c` pour chaque `c` dans `colorations_1_nmoins1` (on peut faire un `List.iter (fun c -> res := (...) :: !res) colorations_1_nmoins1`).
    4. Cette approche construit une liste de colorations représentées comme des listes d'entiers (`int list`), et pas des tableaux d'entiers (`int array`), donc on peut appeler la première fonction `toutes_les_colorations_aux` (récursive), et une deuxième fonction `toutes_les_colorations` qui se contente de convertir le résultat avec `List.map (Array.of_list) (toutes_les_colorations_aux n k)`, car la fonction `Array.of_list` transforme une `'a list` en `'a array`.
11. Proposer un algorithme naïf pour décider si un graphe donné $G = (S, A)$ est k-coloriable (pour k quelconque donné), procédant par une simple exploration combinatoire exhaustive de toutes les k-colorations possibles, en utilisant `toutes_les_colorations` de la question 10. précédente. Implémenter le en une fonction `est_k_coloriable (g:graphe) (k:int) : coloration option` qui renvoie `None` si le graphe n'est pas k-coloriable (aucune des colorations obtenues par `toutes_les_colorations n k` n'était valide), ou `Some c` si la coloration `c` est valide pour le graphe.

12. Tester si le graphe `g1` de l'exemple est 1-coloriable, 2-coloriable, 3-coloriable, et ainsi de suite jusqu'à trouver le k le plus petit tel que `g1` soit k-coloriable.
13. En utilisant cette fonction `est_k_coloriable`, expliquer comment trouver le k le plus petit tel quel G soit k-coloriable. Pourquoi cet algorithme termine ? Quelle peut être la valeur maximum du k trouvé (en fonction de $n = |S|$ le nombre de sommets) ? Implémenter cela en une fonction `trouve_k_min (g:graphe) : (int * coloration)` qui renvoie le couple `(kmin, coloration_k_couleurs)` avec `kmin` le nombre minimal k tel que G soit k-coloriable, et `coloration_k_couleurs` une coloration valide à k couleurs pour ce graphe.
14. Tester `trouve_k_min` sur le graphe `g1` de l'exemple.