Cette *vingtième* colle vous fera écrire des fonctions sur des graphes représentés par des listes d'adjacence en OCaml, des calculs sur des entiers représentés en binaire et d'autres bases, et quelques fonctions très courtes sur des chaînes de caractères en C.

On travaillera depuis la machine virtuelle ClefAgreg2019, et on compilera les fichiers écrits avant d'exécuter les binaires produits.

Ex.1 Parcours en largeur et en profondeur des graphes orientés représentés par listes d'adjacence - OCaml (35 minutes)

On considère dans cet exercice 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.

Quelle est la taille en mémoire de cette représentation d'un graphe G, en fonction de $n = |S|$ et $m = |A|$?

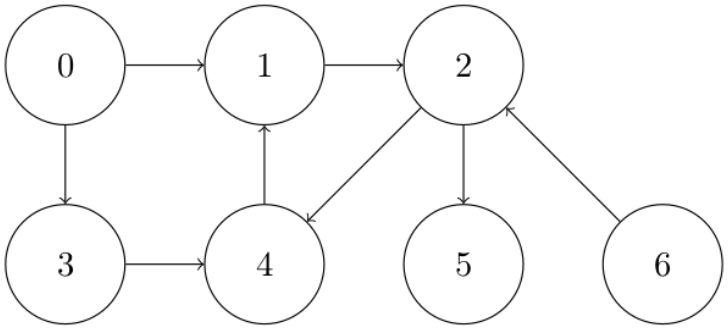

Figure 1 – Un petit graphe G_1.

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).
5. Recopier le code suivant qui implémente le parcours en profondeur, avec non pas une fonction récursive mais une *pile* de sommets à traiter. Pour cette manipuler pile, nous utiliser le module `Stack` de OCaml (*stack* = pile en anglais, en opposition à *queue* = file) :
   - `let pile = Stack.create ()` crée une pile vide,
   - `Stack.is_empty pile` teste si la pile est vide,
   - `let x = Stack.pop pile` extraie le sommet suivant `x` à explorer,
   - `Stack.push y` empile le successeur `y` de `x` à explorer plus tard.

— Pour les calculs de complexité (temporelle), il suffit de savoir que chacune de ses quatre opérations se fait en temps constant $\Theta(1)$.

```
let parcours_profondeur g s =
  let n = nombre_sommets g in
  let vus = Array.make n false in

  let pile = Stack.create () in
  Stack.push s pile;
  Printf.printf "-> Empilement de s = %i\n" s;

  while not (Stack.is_empty pile) do
    let x = Stack.pop pile in
    Printf.printf "<- Dépilement de x = %i\n" x;
    vus.(x) <- true;
    let successeurs_x = successeurs g x in
    let degre_sortant_x = Array.length successeurs_x in
    for i = 0 to degre_sortant_x-1 do
      let y = successeurs_x.(i) in
      if not vus.(y) then begin
        (* y à explorer plus tard *)
        Stack.push y pile;
        Printf.printf "-> Empilement de y = %i\n" y;
      end;
    done;
  done;

  (* On renvoie le tableau de booléens vus *)
  vus
;;
```

6. Justifier sa terminaison.
7. Quelle est sa complexité mémoire, en fonction de $n = |S|$ et $m = |A|$?
8. Quelle est sa complexité temporelle, en fonction de $n = |S|$ et $m = |A|$?
9. Au brouillon, quel sera le contenu du tableau de booléens `vus` renvoyés par `parcours_profondeur` sur le graphe `g1` en partant du sommet `0`. Vérifier avec le code.
10. Implémenter aussi le parcours en largeur, pour lequel il suffit de recopier le code du parcours en profondeur en remplaçant les occurrences du nom du module `Stack` et de la variable `pile` par le module `Queue` et une variable `file`.
11. Tester aussi le parcours en largeur sur le graphe exemple `g1` en partant de différents sommets.

Ex.2 Quelques calculs sur les représentations de nombres entiers (15 minutes)

Au brouillon, répondre aux questions suivantes.

1. Convertir le nombre 2022_{10} en base 2 (binaire), en hexadécimal et en base 8 (octale). Pour convertir en base $8 = 2^3$, on partira de l'écriture en base 2 et on regroupera (en partant de la droite, i.e., des bits de poids faibles) par groupes de trois bits. Idem pour convertir en base $16 = 2^4$ on regroupe par blocs de quatre bits.
   - Par exemple par blocs de trois bits : $110\ 001_2 = XY_8$ où X_8 est $110_2 = 6$ et Y_8 est $001_2 = 1$, donc $49_{10} = 110\ 001_2 = 61_8 = 6 * 8^1 + 1 * 8^0 = 6 * 8 + 1 = 48 + 1 = 49$.
   - Et par par blocs de quatre bits : $11\ 0001_2 = ZW_{16}$ où Z_{16} est $11_2 = 3$ et W_{16} est $0001_2 = 1$, donc $49_{10} = 11\ 0001_2 = 31_{16} = 3 * 16^1 + 1 * 16^0 = 3 * 16 + 1 = 48 + 1 = 49$.
2. Calculer, en la posant, $1111\ 0101_2 + 1001\ 1100_2$. Vérifier en convertissant en décimal.
3. Calculer, en la posant, $1111\ 0101_2 \times 1001_2$. Vérifier en convertissant en décimal.

