Projet *À remettre au plus tard le 4 mars à 23h55*

Le projet doit être écrit en utilisant la portion de Ocaml étudiée en cours. Vous pouvez, sans les redéfinir, utiliser les fonctions prédéfinies sur les listes `mem`, `map`, `fold_left` et `right`, `find`, `find_all`, `exists` et `for_all`.

Le but du projet est de rechercher les composantes fortement connexes (CFC) dans un graphe orienté. On utilisera pour cela deux algorithmes distincts. Le premier effectue deux parcours en profondeur, l'un du graphe et l'autre de son inverse. Le second, l'algorithme de Tarjan, effectue un seul parcours en profondeur (mais plus complexe). Ces algorithmes vous sont rappelés (en mode impératif) dans le document joint extrait du livre « Types de données et algorithmes » de Froidevaux, Gaudel & Soria (1994).

Un graphe sera représenté par une liste de couples **(s, ls)** où **s** est un sommet (type **int**) et **ls** la liste de ses successeurs (type **int list**).

Consignes

Vous devez déposer sur Moodle le ou les fichiers contenant votre projet. Il pourra contenir une brève présentation de votre travail : ce qui est fait (ou non), des choix que vous avez effectués le cas échéant, ou tout commentaire pertinent.

Chaque fonction doit être commentée (son rôle, ses arguments, son résultat et toute explication qui peut aider la lecture et la compréhension de votre travail). Les noms des fonctions et autres identificateurs doivent également permettre de faciliter la lecture.

Il sera apprécié que vous utilisiez des fonctions d'ordre supérieur dès que cela est pertinent.

Algo 1

Vous devez écrire des fonctions permettant d'effectuer les opérations suivantes sur un graphe :

1. Retourner la liste de tous les sommets.
2. Retourner la liste des successeurs d'un sommet donné.
3. Inverser un graphe.
4. Effectuer le parcours en profondeur d'un graphe et retourner la liste des sommets parcourus en ordre suffixe (inversé).
5. Rechercher les composantes fortement connexes par parcours du graphe et de son inverse.

```
# let graphe1 = [(1,[6;7;8]) ; (2,[1;4]) ; (3, [2]) ; (4, [3;5]) ; (5, [1])
; (6, [5;7]) ; (7, []) ; (8, [6;7])];;
 val graphe1 : (int * int list) list =  [(1, [6; 7; 8]); (2, [1; 4]);
(3, [2]); (4, [3; 5]); (5, [1]); (6, [5; 7]); (7, []); (8, [6; 7])]

# liste_succ2 4 graphe1;;
 - : int list = [3; 5]

# inverse_graphe graphe1;;
 - : (int * int list) list =[(1, [2; 5]); (2, [3]); (3, [4]); (4, [2]);
(5, [4; 6]); (6, [1; 8]); (7, [1; 6; 8]); (8, [1])]

# parcours_prof graphe1;;
 - : int list = [2; 4; 3; 1; 8; 6; 7; 5]
(* pour retrouver l'ordre suffixe, lire la liste à partir de la droite *)

# connexites graphe1;;
 - : int list list = [[7]; [8; 6; 5; 1]; [4; 3; 2]]
```

Algo 2 (Tarjan)

Vous devez écrire les fonctions permettant d'implémenter l'algorithme de Tarjan. Pour cela n'hésitez pas à définir des fonctions intermédiaires (bien documentées) permettant de clarifier votre code. En particulier :

- A chaque sommet sont associés deux numéros : le numéro du sommet lors du parcours en profondeur du graphe, dans l'ordre préfixe, et le plus petit numéro préfixe d'un sommet accessible. Précisez comment vous représentez ces informations et écrivez des fonctions permettant d'accéder à ces numéros et de les modifier.
- Écrivez également les fonctions permettant d'empiler un sommet sur une pile et de dépiler tous les sommets jusqu'à un sommet donné (cette dernière fonction pourra retourner un couple constitué des sommets dépilés et de la pile restante).