Partie I : représentations des graphes

On considère des graphes orientés représentés par des listes d'adjacence dont les sommets sont des entiers indexés de 0 à $n-1$, n étant le nombre de sommets du graphe. Un tel graphe sera représenté par un tableau `g` de listes tel que `g.(i)` sera la liste des voisins du sommet `i`. On prendra garde à ne pas contenir plusieurs fois un même sommet au sein d'une liste. On utilisera le type suivant en Ocaml :
`type graphe = int list array;;`

Question 1
Ecrire une question `graphe_vide : int -> graphe` qui crée un graphe à n sommets sans arcs.

Question 2
Ecrire une fonction `ajoute_arc : graphe -> int -> int -> unit` tel que l'appel `ajoute_arc g i j` ajoute l'arc (i, j) au graphe g. Si l'arc est déjà présent, la fonction ne fera rien

Question 3
Ecrire une fonction `supprime_arc : graphe -> int -> int -> unit` qui supprime un arc d'un graphe. Cette fonction ne fera rien si l'arc que l'on veut supprimer n'est pas dans le graphe.

Un graphe est non orienté si, pour tout $(i, j) \in A$, $(j, i) \in A$.

Question 4
Ecrire une fonction `non_oriente : graphe -> bool` qui vérifie si un graphe est non orienté.

Le graphe transposé d'un graphe $G = (S, A)$ est le graphe $G^\top = (S, A^\top)$ tel que $A^\top\{(i, j) \in S^2 \,/\, (j, i) \in A\}$.

Question 5
Ecrire une fonction `transpose : graphe -> graphe` qui prend en argument un graphe et renvoie son graphe transposé.

Partie II : parcours en largeur, composantes connexes

Dans tout ce paragraphe, les parcours de graphe attendus sont des parcours en largeur.

Question 6
Ecrire une fonction `sommets_accessibles : graphe -> int -> int list` qui renvoie la liste des sommets accessibles depuis un sommet au sein d'un graphe.

Un chemin sera représenté par une liste d'entiers.

Question 7
Ecrire une fonction `chemins : graphe -> int -> int list array` qui, étant donné un graphe g et un sommet i renvoie le tableau c tel que, pour tout j, $c.(j)$ est un chemin de i à j (si un tel chemin n'existe pas, $c.(j)$ sera le chemin vide).
Un graphe non orienté est connexe s'il existe un chemin entre tout couple de sommets.

Question 8
Ecrire une fonction `connexe : graphe -> bool` qui vérifie si un graphe, supposé non orienté, est connexe.

La composante connexe d'un sommet dans un graphe non orienté est l'ensemble des sommets accessibles depuis ce sommet (on peut vérifier que la relation "il existe un chemin d'un sommet à un autre" est une relation d'équivalence

sur l'ensemble des sommets ; ses classes d'équivalence sont les composantes connexes).

Question 9
Ecrire une fonction `comp_connexes : graphe -> int list list` qui renvoie la liste des composantes connexes d'un graphe non orienté. Une composante connexe sera renvoyé sous la forme d'une liste de sommets.

Dans un graphe orienté, la relation définie par, pour tout $(i, j) \in S^2$, "il existe un chemin de i à j et de j à i" est une relation d'équivalence. Ses classes d'équivalence sont appelées les composantes fortement connexes du graphe.
On appelle sommets co-accessibles depuis $i \in S$ l'ensemble des sommets $j \in S$ tels qu'il existe un chemin de j à i.

Question 10
Ecrire une fonction `co_accessibles : graphe -> int -> int list` qui renvoie la liste des sommets co-accessibles depuis un sommet dans un graphe.

Question 11
Ecrire une fonction `comp_fort_connexe : graphe -> int -> int list` qui renvoie la composante fortement connexe d'un sommet dans un graphe.

Question 12
Ecrire une fonction `comp_fort_connexes : graphe -> int -> int list list` qui renvoie la liste des composantes fortement connexes d'un graphe.

Partie III : parcours en profondeur, cycles

Le parcours en profondeur peut aussi être utiliser pour chercher des sommets accessibles et des composantes connexes. On l'utilisera dans la suite pour chercher des cycle.

Un cycle au sein d'un graphe est un chemin partant d'un sommet revenant à ce sommet. Un graphe est dit cyclique s'il admet un cycle.

Question 13
En effectuant un parcours en profondeur, écrire une fonction `cycle_depuis_sommet : int -> graphe -> bool` qui, étant donné un sommet s, un graphe g, vérifie s'il existe un cycle partant de s et revenant en s.

Question 14
Ecrire une fonction `cyclique : graphe -> bool` qui vérifie si un graphe possède un cycle.

Question 15
Ecrire une fonction `cycle : graphe -> int list` qui, étant donné un graphe cyclique, renvoie un cycle. On ne vérifiera pas s'il admet bien un cycle.

Dans un graphe non orienté, toute arête $\{i, j\}$ permet de trouver un cycle trivial : le cycle (i, j, i). On appellera cycle dans un graphe non orienté $G = (S, A)$ tout chemin $(s_0, \cdots, s_n) \in S^{n+1}$ tel que, pour tout $0 \leq i \leq n-1$, $\{s_i, s_{i+1}\} \in A$ et, pour tout $0 \leq i \leq n-2$, $s_i \neq s_{i+2}$.

Question 16
Ecrire des fonctions `cyclique_non_oriente : graphe -> bool` qui, étant donné un graphe non orienté, vérifie si ce graphe admet un cycle et `cycle_non_oriente : graphe -> int list` qui, étant donné un graphe non orienté cyclique, renvoie un cycle.