TD 2 : Graphes, parcours de graphes

*Les questions de programmation seront obligatoirement **vérifiées** sur machine.*

Exercice 1

Soit $G = (S, A)$ graphe orienté. On appelle *graphe miroir* de G le graphe orienté $\overline{G} = (S, \overline{A})$ où $\overline{A} = \{(y, x) \ / \ (x, y) \in A\}$.

1. Si M est la matrice d'adjacence de G, que dire de la matrice d'adjacence de $\overline{G}$?
2. Soit T la représentation par listes d'adjacence de G, écrire un algorithme qui cosntruit T' la représenation par listes d'adjacence de $\overline{G}$.
3. Écrire une fonction Caml `transpose : int list array -> int list array` implémentant l'algorithme précédent.
4. Écrire cette même fonction en langage C. On supposera que le prototype de la fonction est

```
void transpose(int n,
               int t[50][100],
               int resultat[50][100])
```

où n est le nombre de sommets (≤ 50) et $t[i][0]$ le degré sortant du sommet numéro i. Les autres cases de la ligne $t[i]$ sont les numéros de voisins sortants.

Exercice 2

Un graphe est-il un arbre ?

Soit $G = (S, A)$ un graphe non orienté.

1. Rappeler ce que signifie G est un arbre.
2. Écrire une fonction C :

```
bool estarbre(int n,
              int t[50][100]);
```

testant si le graphe G à $n \leq 50$ sommets donné sous forme de listes d'adjacence t est un arbre. Dans le tableau à deux entrées t, la case $t[i][0]$ contient le degré du sommet numéro i et les autres cases de la ligne $t[i]$ sont les numéros de voisins du sommet i.

Exercice 3

2-colorabilité d'un graphe

Soit $G = (S, A)$ un graphe non orienté. On dit que G est 2-colorable s'il existe une application

$$\nu : S \to \{0, 1\}$$

telle que $\forall (x, y) \in S^2,\ \{x, y\} \in A \Rightarrow \nu(x) \neq \nu(y)$. Cet exercice a pour but de démontrer le résultat : un graphe est 2 colorable si et seulement si tous ses cycles sont de longueur paire.

1. Rappeler les définitions de chemin, cycle et longueur d'un chemin/cycle.
2. Démontrer que si G est 2-colorable alors tous ses cyles sont de longueur paire.
3. Réciproquement, soit G un graphe dont tous les cycles sont de longueur paire, montrer que l'on peut déterminer une coloration ν de ses sommets. Pour cela on écrira une fonction C :

```
void colore2(int n,
             int t[50][100],
             couleurs c[50])
```

reposant sur un parcours en profondeur itératif qui construit une coloration adaptée. Dans les arguments t est la représentation du graphe sous forme de listes d'adjacence où $n \leq 50$ est le nombre de sommets, $t[i][0]$ le degré du sommet numéro i. Les autres cases de la ligne $t[i]$ sont les numéros de voisins.

4. Écrire la même fonction, mais de manière récursive en OCaml :

```
colore2 : int list array -> int array
```

5. Donner la complexité de l'algorithme des questions 4 et 5 en fonction de $n = \text{Card}(S)$ et $m = \text{Card(A)}$.
6. Un arbre est-il 2-colorable ?

Exercice 4

Composantes connexes

Soit $G = (S, A)$ un graphe non-orienté.

1. Rappeler la définition de *composante connexe* de G.
2. Écrire une fonction C :

```
int nb_composantes(int n,
                   int t[50][100]);
```

comptant le nombre de composantes connexes de G. Le graphe G est donné sous forme de listes d'adjacence où $n \leq 50$ est le nombre de sommets, la case $t[i][0]$ contient le degré du sommet numéro i et les autres cases de la ligne $t[i]$ sont les numéros de voisins du sommet i.

3. Écrire une fonction C :

```
int taille_composante_max(int n,
                          int t[50][100]);
```

avec les mêmes conventions que la question précédente pour les arguments.

⋆Exercice 5

Grenouilles et nénuphars

Deux familles de grenouilles composées respectivement de p et q grenouilles sont situées sur deux rives opposées d'un lac. Il est possible de traverser le lac à l'aide d'une rangée de $n > p + q$ nénuphars parfaitement alignés. Initialement, les grenouilles de la première famille sont situées sur les p nénuphars les plus à gauche et ceux de la seconde famille sur les q nénuphars les plus à droite. Les grenouilles sont civilisées et obéissent aux règles de comportement suivantes :

- ⋆ elles avancent toujours (vers la droite pour la famille 1 et la gauche pour la famille 2),
- ⋆ elles peuvent faire des bonds de distance 1 ou 2 nénuphars,
- ⋆ une seule grenouille saute à la fois,
- ⋆ une grenouille ne saute jamais sur un nenuphar déjà occupé par une autre grenouille.

Le but est que les grenouilles de la famille 1 arrivent sur les p nénuphars les plus à droite et que celles de la famille 2 arrivent sur les q nénuphars les plus à gauche.

1. Proposer un type Caml `etat` représentant une situation actuelle d'occupation des nenuphars.
2. Ecrire une fonction `successeurs : etat -> etat list` qui calcule la liste des etats qu'on peut atteindre après 1 saut de grenouille.
3. Ecrire une fonction `optimal : int -> int -> int -> int` telle que `optimal p q n` retourne le nombre optimal (minimum) de sauts de grenouilles pour réussir la double traversée ou -1 si le problème est insoluble.