TP6 - CFC et 2SAT

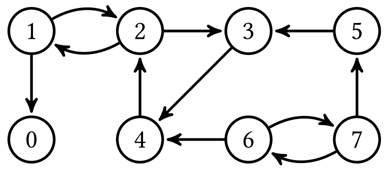
Le langage de programmation est OCaml.

Graphes

On considère des graphes *orientés non pondérés* dont les sommets sont étiquetés par des entiers naturels consécutifs à partir de 0. Les sommets d'un graphe d'ordre n sont étiquetés par les entiers $0, 1, \dots, n-1$. On implémente les graphes par des listes d'adjacences et on définit le type suivant.

```
type graph = int list array
```

Question 1. Définir l'identificateur `g` associé au graphe représenté ci-dessous.



Question 2. Écrire une fonction `succ : graph -> int -> int list` telle que `succ g v` renvoie la liste des sommets adjacents au sommet `v` du graphe `g`.

Question 3. Écrire une fonction `dfs : graph -> int -> bool array` telle que `dfs g src` réalise un parcours en profondeur du graphe `g` à partir du sommet `src` et renvoie un tableau de booléens de taille l'ordre du graphe dont les éléments sont à `true` si un sommet d'indice son étiquette a été visité, à `false` sinon. Par exemple, `dfs g 3` renvoie : `[|true; true; true; true; true; false; false; false|]`.

Question 4. Écrire une fonction `mirror : graph -> graph` telle que `mirror g` renvoie le graphe miroir de `g`, c'est-à-dire le graphe ayant les mêmes sommet que `g` et dont les arcs sont de sens inverses de deux de `g`.

Question 5. La fonction `post_order` suivante renvoie une liste des sommets visités lors du parcours en profondeur d'un graphe, dans l'*ordre postfixe*. Sur le document réponse, détailler les étapes de construction de cette liste pour le graphe exemple `g`.

```
let post_order g =
  let n = size g in
  let visited = Array.make n false in
  let order = ref [] in
  let rec dfs v =
    if not visited.(v) then (
      visited.(v) <- true;
      List.iter dfs (succ g v);
      order := v :: !order
    )
  in
  for v = 0 to n-1 do dfs v done;
  !order;;
```

Composantes fortement connexes

Question 6. Rappeler ce que sont les *composantes fortement connexes* (CFC) d'un graphe orienté.

Question 7. Quelles sont celles du graphe `g` ?

Question 8. On se propose de calculer les composantes fortement connexes sous la forme d'un tableau donnant, pour chaque sommet, le numéro de sa composante fortement connexe. S'il y en a N, elles sont numérotées $0, 1, \dots, N-1$, dans un ordre arbitraire. Donner un tel tableau pour le graphe `g`.

Question 9. L'*algorithme de Kosaraju–Sharir* construit ce tableau en deux temps.

- Un premier parcours en profondeur visite tous les sommets du graphe miroir du graphe initial et construit une liste des sommets visités dans l'ordre postfixe.
- Un second parcours en profondeur visite tous les sommets du graphe en suivant l'ordre des sommets de la liste obtenue à l'étape précédente. Le tableau des composantes fortement connexes défini à la question précédente est construit lors de cette seconde phase.

Écrire une fonction `kosaraju_sharir : graph -> int * int array` telle que `kosaraju_sharir g` renvoie un couple formé du nombre de composantes fortement connexes du graphe `g` et le tableau dont la construction est décrite ci-dessus.

2-SAT

On considère des formules logiques sous forme de 2-CNF, *forme normale conjonctive* (*Conjonctive Normal Form* en anglais) où chaque clause comporte, au plus, deux littéraux. Un algorithme de complexité temporelle polynomiale établit la satisfiabilité du problème 2-SAT en transformant une instance 2-SAT en l'analyse des composantes fortement connexes d'un graphe. Le principe de cette *transformation* repose sur l'observation que toute clause $(l_i \vee l_j)$ est logiquement équivalente à $(\neg l_i) \rightarrow l_j$, elle même équivalente à sa contraposée $(\neg l_j) \rightarrow l_i$. Une clause formée d'un seul littéral l_i est équivalente à $(l_i \vee l_i)$ puis à $(\neg l_i) \rightarrow l_i$, qui est sa propre contraposée.

Soit φ une 2-CNF comportant $n \in \mathbb{N}^*$ variables propositionnelles désignées par x_i, $i \in [\![0, n-1]\!]$. La procédure suivante construit un graphe orienté G à partir de φ.

- À chaque variable x_i, on associe deux sommets v_{2i} et v_{2i+1} dans G qui représentent respectivement le littéral x_i et le littéral $\neg x_i$.
- Pour chaque clause de type (l_i), G comporte un arc, soit du sommet v_{2i+1} au sommet v_{2i} si $l_i = x_i$, soit du sommet v_{2i} au sommet v_{2i+1} si $l_i = \neg x_i$.
- Pour chaque clause du type $(l_i \vee l_j)$ où les littéraux l_i et l_j contiennent des variables x_i ,x_j distinctes, G comporte deux arêtes choisies en fonction des cas suivants :
  - si $l_i = x_i$ et $l_j = x_j$ alors on ajoute les arêtes (v_{2i+1}, v_{2j}) et (v_{2j+1}, v_{2i}),
  - si $l_i = x_i$ et $l_j = \neg x_j$ alors on ajoute les arêtes (v_{2i+1}, v_{2j+1}) et (v_{2j}, v_{2i}),
  - si $l_i = \neg x_i$ et $l_j = x_j$ alors on ajoute les arêtes (v_{2i}, v_{2j}) et (v_{2j+1}, v_{2i+1}),
  - si $l_i = \neg x_i$ et $l_j = \neg x_j$ alors on ajoute les arêtes (v_{2i}, v_{2j+1}) et (v_{2j}, v_{2i+1}).
- Pour chaque clause du type $(l_i \vee l_j)$ où les littéraux l_i, l_j contiennent la même variable $x_i = x_j$, soit on élimine directement la clause si elle est de la forme $(x_i \vee \neg x_i)$ ou $(\neg x_i \vee x_i)$, soit on se ramène au cas d'une clause (l_i).

On définit les types suivants.

```
(* variable ou négation de variable *)
type litteral = V of int | NV of int
type clause = litteral list
type fnc = clause list
```

Question 10. Constuire le *graphe d'implication* de la formule suivante : $\varphi = (x_0 \vee \neg x_1) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_0)$.

Question 11. Écrire une fonction `var_max : fnc -> int` qui prend en entrée une CNF φ et renvoie le plus grand indice de variable utilisé dans la formule.

Question 12. Écrire une fonction `sat2graph : fnc -> graph` telle `sat2graph phi` renvoie le graphe associé à la 2-CNF représentée par `phi` en suivant la procédure précédente.
Pour simplifier, on suppose qu'aucune clause n'est répétée dans φ et qu'aucune clause n'est de type $(l_i \vee l_j)$ où l_i, l_j contiennent la même variable $x_i = x_j$.

Question 13. On admet qu'une 2-CNF φ est satisfiable si et seulement si il n'existe pas de variable x_i dont les deux sommets correspondants v_{2i} et v_{2i+1} sont dans la même composante fortement connexe du graphe G associé.
□ **13.1.** Écrire une fonction `sat2 : fnc -> bool` telle que `sat2 phi` renvoie un booléen indiquant si la 2-CNF représentée par `phi` est satisfiable ou non.
□ **13.2.** Estimer sa complexité.