Exercice 1

On fixe $n \in \mathbb{N}^*$. Soient $\varphi : [\![1,n]\!] \to [\![1,n]\!]$ et $\psi : [\![1,n]\!] \to [\![1,n]\!]$. Soient u et v deux éléments de $[\![1,n]\!]$. On dit que u et v sont (φ,ψ)-équivalents s'il existe $k \in \mathbb{N}$ et un $(k+2)$-uplet $(w_0, w_1, \ldots, w_{k+1}) \in [\![1,n]\!]^{k+2}$ avec $w_0 = u$, $w_{k+1} = v$ vérifiant :

$$\forall i \in [\![0,k]\!], \varphi(w_i) = \varphi(w_{i+1}) \text{ ou } \psi(w_i) = \psi(w_{i+1})$$

L'objectif est d'écrire un algorithme en pseudo-code permettant de calculer les différentes classes d'équivalence engendrées par cette relation.

1. Justifier rapidement que « être (φ,ψ)-équivalent » est une relation d'équivalence sur l'ensemble $[\![1,n]\!]$.
2. Pour cette question, on considère les applications φ et ψ définies par :

$i =$	1	2	3	4	5	6	7	8	9
$\varphi(i) =$	3	2	2	9	6	4	9	5	7
$\psi(i) =$	5	1	3	4	5	1	7	7	4

   Calculer les différentes classes d'équivalences.
3. On revient au cas général. On définit le graphe $G = (S, A)$ par :

$$S = [\![1,n]\!], A = \{(x,y) \in S^2 \mid x \neq y \text{ et } (\varphi(x) = \varphi(y) \text{ ou } \psi(x) = \psi(y))\}$$

   On fixe x et y deux sommets différents dans S. Traduire sur le graphe G le fait que les sommets x et y sont (φ,ψ)-équivalents et en déduire que le calcul des classes d'équivalence de G se traduit en un problème classique sur les graphes que l'on précisera.
4. Donner en pseudo-code un algorithme permettant de résoudre le problème correspondant sur les graphes.

On fixe n un nombre pair. On considère deux applications φ et ψ de $[\![1,n]\!]$ où tout élément de l'image de φ admet exactement deux antécédents par φ et où tout élément de l'image de ψ admet exactement deux antécédents par ψ. Pour $f \in \{\varphi,\psi\}$, on note G_f le graphe $(S, \{(x,y) \in S^2 \mid x \neq y \text{ et } f(x) = f(y)\})$.

5. Préciser la forme du graphe G_f pour $f \in \{\varphi,\psi\}$.
6. Expliciter la forme des différentes classes d'équivalence dans le graphe G correspondant.

Exercice 2

On attend un style de programmation fonctionnel. L'utilisation des fonctions du module `List` *est autorisée ; celle des fonctions du module* `Option` *est interdite.*

Une formule du calcul propositionnel est une *formule de Horn* s'il s'agit d'une formule sous forme normale conjonctive (FNC) dans laquelle chaque clause (éventuellement vide, auquel cas la clause en question est la disjonction d'un ensemble vide de littéraux et est donc sémantiquement équivalente à $\bot$) contient au plus un littéral positif. Dans la suite, on considère qu'une clause d'une telle formule contient au plus une occurrence de chaque variable (en particulier, les clauses sont sans doublons).

1. Les formules suivantes sont-elles des formules de Horn ?
   (a) $F_1 = (\neg x_0 \vee \neg x_1 \vee \neg x_3) \wedge (x_0 \vee \neg x_1) \wedge (\neg x_2 \vee x_0 \vee \neg x_3) \wedge (\neg x_0 \vee \neg x_3 \vee x_2) \wedge x_2 \wedge (\neg x_3 \vee \neg x_2)$.
   (b) $F_2 = (x_0 \vee \neg x_1) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_0) \wedge \neg x_1 \wedge (x_1 \vee \neg x_1 \vee x_0) \wedge (\neg x_0 \vee x_2)$.
   (c) $F_3 = (\neg x_1 \vee \neg x_4) \wedge x_1 \wedge (\neg x_0 \vee \neg x_3 \vee \neg x_4) \wedge (x_0 \vee \neg x_1) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_4 \vee \neg x_0 \vee \neg x_1)$.

On utilise le type suivant pour manipuler les formules de Horn : une formule de Horn est une liste de clauses de Horn ; une clause étant la donnée d'un `int option`, valant `None` si la clause ne contient pas de littéral positif et `Some i` si x_i en est l'unique littéral positif, et d'une liste d'entiers correspondant

aux numéros des variables intervenant dans les littéraux négatifs.

```
type clause_horn = int option * int list
type formule_horn = clause_horn list
```

2. Écrire une fonction `avoir_clause_vide : formule_horn -> bool` qui renvoie `true` si et seulement si la formule en entrée contient un clause vide (donc ne contenant aucun littéral positif ou négatif).

On appelle *clause unitaire* une clause réduite à un littéral positif. Par ailleurs, *propager* une variable x_i dans une formule F sous FNC consiste à modifier F comme suit :

- Toute clause de F qui ne fait pas intervenir la variable x_i est conservée telle quelle.
- Toute clause de F qui fait intervenir le littéral x_i est supprimée entièrement.
- On supprime le littéral $\neg x_i$ de toutes les clauses de F qui font intervenir ce littéral.

On souligne que supprimer $\neg x_i$ d'une clause C qui ne fait intervenir que ce littéral ne revient pas à supprimer la clause C. On s'intéresse à l'algorithme $\mathcal{A}$ suivant dont on admet (pour le moment) qu'il permet de déterminer si une formule de Horn F est satisfiable.

Début algorithme
  Tant que il y a une clause unitaire x_i dans F **Faire**
    $F \leftarrow$ propager x_i dans F
  Si F contient une clause vide **Alors**
    Renvoyer Faux
  Sinon
    Renvoyer Vrai

1. À l'aide de cet algorithme, déterminer si les formules de Horn de la question 1 sont satisfiables. On utilisera ces formules pour tester les fonctions implémentées aux questions suivantes.
2. Écrire une fonction `trouver_clause_unitaire : formule_horn -> int option` renvoyant `None` si la formule en entrée n'a pas de clause unitaire et `Some i` où x_i est l'une des clauses unitaires sinon.
3. Justifier que propager une variable dans une formule de Horn donne une formule de Horn. Écrire une fonction `propager : formule_horn -> int -> formule_horn` qui prend en entrée une formule de Horn F et un entier i et calcule la formule résultat de la propagation de x_i dans F.
4. Déduire des questions précédentes une fonction `etre_satisfiable : formule_horn -> bool` renvoyant `true` si et seulement si la formule de Horn en entrée est satisfiable.
5. Quelle est la complexité de votre algorithme en fonction de la taille de la formule en entrée ? Que peut-on dire des problèmes de décision `SAT` et `HORN-SAT` (dont la définition est la même que celle de `SAT` à ceci près que les formules considérées sont supposées être des formules de Horn) ?
6. On s'intéresse à présent à la correction de l'algorithme $\mathcal{A}$.
   (a) Si F est une clause de Horn sans clause unitaire, ni clause vide, donner une valuation simple qui satisfait F.
   (b) On admet que si F est une formule de Horn faisant intervenir une clause unitaire x_i et F' est le résultat de la propagation de x_i dans F, alors F est satisfiable si et seulement si F' est satisfiable. En déduire la correction de l'algorithme $\mathcal{A}$.
   (c) Expliquer comment modifier les fonctions précédentes afin de déterminer une valuation satisfaisant une formule de Horn dans le cas où elle existe plutôt que de juste dire si elle est satisfiable ou non. On ne demande pas d'implémentation.