On s'intéresse dans ce TP à la résolution du problème `MAXSAT` :

* **Instance** : une formule booléenne φ en forme normale conjonctive sur $\mathcal{V} = \{x_0, \dots, x_{n-1}\}$ contenant m clauses disjonctives $C_1, \dots, C_m$.
* **Solution** : une valuation $\mu \in \{0,1\}^{\mathcal{V}}$.
* **Optimisation** : maximiser le nombre de clauses satisfaites, c'est-à-dire le cardinal de l'ensemble $J = \{j \in [\![1, m]\!] \mid \mu(C_j) = 1\}$.

On représente en OCaml une formule en FNC sur $\mathcal{V} = \{x_0, \dots, x_{n-1}\}$ par le type suivant :

```
type litteral = Var of int | Neg of int

type clause = litteral list

type fnc = clause list
```

où un littéral sera représenté par un objet `litteral`, une liste de littéraux sera interprétée comme une clause disjonctive de ces littéraux, et une liste de clauses disjonctives sera interprétée comme une FNC.

Par exemple, la formule $\varphi = (\overline{x_0} \vee x_1 \vee \overline{x_2}) \wedge (x_0 \vee \overline{x_1} \vee x_2) \wedge (x_0 \vee x_1 \vee x_2) \wedge (\overline{x_0} \vee \overline{x_1})$ sera représentée par :

```
let phi = [[Neg 0; Var 1; Neg 2];
           [Var 0; Neg 1; Var 2];
           [Var 0; Var 1; Var 2];
           [Neg 0; Neg 1]]
```

Par ailleurs, on représentera une valuation sur $\mathcal{V} = \{x_0, \dots, x_{n-1}\}$ par un tableau d'entiers de taille n, contenant des 0 et des 1, tel que si $\mu \in \{0,1\}^{\mathcal{V}}$ est représenté par `mu`, alors `mu.(i)` $= \mu(x_i)$ pour tout $i \in [\![0, n-1]\!]$.

Exercice 1

On veut générer des formules de manière pseudo-aléatoire. Étant donné $u_0 \in \mathbb{N}$, on définit par récurrence la suite :

$$\forall i \in \mathbb{N}, u_{i+1} = 19\,999\,999 \times u_i \pmod{19\,999\,981}$$

Pour toute la suite, on considère que la graine vaut $u_0 = 42$, mais utiliser une autre graine pourrait permettre de générer d'autres formules.

1. Vérifier que les valeurs de u_i modulo 1000, pour $i = 123$, $i = 456\,000$ et $i = 789\,000\,000$, sont 768, 915 et 229.
2. Vérifier que les valeurs de $\sum_{i=0}^{m} u_i$ modulo 1000, pour $m = 123$, $m = 45\,600$ et $m = 78\,900\,000$, sont 543, 75 et 665.

   Pour ce genre de question, il est largement préférable d'écrire une fonction qui calcule le tableau des $(u_i)_{i \leqslant n}$ plutôt qu'une fonction qui calcule u_i. On peut même choisir de remplir un tableau global une seule fois, qu'on utilise pour la suite.

Pour $i \in \mathbb{N}$, $n \in \mathbb{N}^*$, $m \in \mathbb{N}$ et $k \in \mathbb{N}^*$, on veut définir la formule $\varphi(i, n, m, k)$ comme une formule à n variables et m clauses, l'indice i permettant de générer plusieurs formules de ce genre et l'indice k permettant de régler la taille moyenne des clauses.

Formellement, on définit la clause $C(i, n, k)$ de la manière suivante : pour $j \in [\![0, n-1]\!]$, on pose $r = u_{i+j} \pmod{2n}$:

- si $r \geqslant k$, alors la clause ne contient ni x_j, ni $\overline{x_j}$;
- sinon, si r est pair, la clause contient x_j, sinon la clause contient $\overline{x_j}$.

3. Écrire une fonction `clause : int -> int -> int -> clause` qui calcule la clause $C(i, n, k)$.

Dès lors, la formule $\varphi(i,n,m,k)$ est la formule :

$$\varphi(i,n,m,k) = \bigwedge_{j=0}^{m-1} C(i+j\times n, n, k)$$

4. Écrire une fonction `formule : int -> int -> int -> int -> clause` qui calcule la formule $\varphi(i,n,m,k)$.
5. Vérifier que la formule $\varphi(0,3,4,2)$ est $(x_0 \vee x_1 \vee x_2) \wedge (x_0 \vee \overline{x_2}) \wedge \overline{x_0} \wedge x_0$.

Palier 1

Exercice 2

1. Écrire une fonction `evaluer : clause -> int array -> int` qui prend en argument une clause disjonctive C et une valuation μ et calcule $\mu(C)$. On supposera sans le vérifier que C et μ sont bien définies sur le même ensemble de variables.
2. En déduire une fonction `taille_J : fnc -> int array -> int` qui prend en argument une formule φ en FNC et une valuation μ et renvoie le nombre de clauses de φ satisfaites par μ. On considère qu'une clause vide n'est pas satisfiable.

On souhaite résoudre le problème `MAXSAT` par une approche par retour sur trace, qu'on adaptera par la suite en un algorithme de type *Branch and Bound*. Pour ce faire, le format d'une solution partielle $\tilde{\mu}$ est une valuation définie sur les variables $\{x_0,\dots,x_{i-1}\}$. Dans l'arbre des possibilités, une telle solution partielle sera complétée avec les enfants attribuant les affectations $\mu(x_i)=0$ et $\mu(x_i)=1$.

3. Écrire une fonction `maxsat_backtracking : fnc -> int -> int array` qui prend en argument une formule φ et l'entier n correspondant au nombre de variables et résout le problème `MAXSAT`.
4. Vérifier que pour les formules $\varphi(0,10,20,3)$, $\varphi(0,15,100,5)$ et $\varphi(0,20,200,7)$, le nombre maximal de clauses satisfaites est 14, 89 et 186 respectivement.
5. Quelle est la complexité d'une telle fonction ?

Exercice 3

On considère l'algorithme qui consiste à choisir aléatoirement et uniformément une valuation et à la renvoyer. Par abus de notation, on note J le cardinal de l'ensemble J.

1. [À faire après le TP] On suppose que chaque clause de φ contient au moins k littéraux indépendants (donc portant sur des variables différentes). Montrer que $\mathbb{E}(J) \geqslant m\left(1-\frac{1}{2^k}\right)$.
2. [À faire après le TP] En déduire que l'algorithme précédent est un algorithme Monte Carlo qui renvoie une valuation qui satisfait au moins la moitié des clauses non vides avec probabilité au moins $\frac{1}{2}$.

Pour $i\in\mathbb{N}$ et $n\in\mathbb{N}^*$, la valuation $\mu(i,n)$ est définie sur n variables par :

$$\forall j\in[\![0,n-1]\!], \mu(i,n)(x_j) = u_{i+j} \pmod 2$$

3. Écrire une fonction `maxsat_alea : int -> int -> int array` qui renvoie la valuation $\mu(i,n)$.
4. Écrire une fonction `simulation : fnc -> int -> int -> float` qui prend en argument une formule, son nombre de variables n et un entier m et fait m simulations du calcul précédent, pour $i\in\{0,n,2n,\dots,(m-1)n\}$, puis renvoie la moyenne du ratio $\frac{J}{J^*}$, J étant l'ensemble de clauses satisfaites par une valuation renvoyée par l'algorithme précédent et J^* un ensemble de clauses satisfaites de taille maximal.

   Indication : on pourra utiliser `float_of_int` pour convertir en flottant
5. Vérifier que pour $m=1000$, sur les formules $\varphi(0,10,20,3)$, $\varphi(0,15,100,5)$ et $\varphi(0,20,200,7)$, une

valuation aléatoire satisfait en moyenne respectivement 76,1%, 83,1% et 88,6% du maximum du nombre de clauses satisfiables.

Palier 2

Malheureusement, cet algorithme probabiliste n'est pas une 2-approximation de `MAXSAT`, car il ne garantit pas que le cardinal de J vérifie la même inégalité que son espérance. On propose de dérandomiser l'algorithme précédent, quitte à augmenter légèrement sa complexité.

Exercice 4

Pour $i \in [\![0, n-1]\!]$, on note μ_i une valuation partielle, c'est-à-dire définie sur $\{x_0, \ldots, x_i\}$ mais non définie sur $\{x_{i+1}, \ldots, x_{n-1}\}$. On note $\mathbb{E}(J \mid \mu_i)$ le nombre moyen de clauses satisfaites en complétant μ_i de manière aléatoire uniforme pour les variables $\{x_{i+1}, \ldots, x_{n-1}\}$ (c'est une espérance conditionnelle). On considère l'algorithme suivant :

Début algorithme
  $\mu \leftarrow$ valuation vide.
  Pour $i = 0$ à $n-1$ **Faire**
    Choisir $\mu(x_i)$ tel que $\mathbb{E}(J \mid \mu_i)$ est maximal.
  Renvoyer μ.

1. [À faire après le TP] Montrer que pour $i \in [\![0, n-2]\!]$:

$$\mathbb{E}(J \mid \mu_i) = \frac{1}{2}\left(\mathbb{E}(J \mid \mu_i, \mu(x_{i+1}) = 1) + \mathbb{E}(J \mid \mu_i, \mu(x_{i+1}) = 0)\right)$$

2. [À faire après le TP] En déduire que la propriété $\mathbb{E}(J \mid \mu_i) \geqslant \mathbb{E}(J)$ est un invariant de boucle dans l'algorithme précédent.
3. [À faire après le TP] En déduire que l'algorithme décrit précédemment est une 2-approximation de `MAXSAT`.

Pour calculer $\mathbb{E}(J \mid \mu_i)$, on remarque que $\mathbb{E}(J \mid \mu_i) = \sum_{j=1}^{m} \mathbb{P}(C_j \mid \mu_i)$ où $\mathbb{P}(C_j \mid \mu_i)$ désigne la probabilité que la clause C_j soit satisfaite si on complète μ_i de manière aléatoire uniforme pour les variables $\{x_{i+1}, \ldots, x_{n-1}\}$. On remarque que :

- si C_j contient un littéral sur une variable $x_0, \ldots x_i$ évalué à vrai, alors $\mathbb{P}(C_j \mid \mu_i) = 1$;
- si C_j ne contient que des littéraux sur des variables $x_0, \ldots, x_i$ évalués à faux, alors $\mathbb{P}(C_j \mid \mu_i) = 0$;
- si C_j ne contient aucun littéral évalué à vrai sur des variables $x_0, \ldots, x_i$ et contient k littéraux sur des variables $x_{i+1}, \ldots, x_{n-1}$, alors $\mathbb{P}(C_j \mid \mu_i) = 1 - \frac{1}{2^k}$.

4. Écrire une fonction `proba_condi : clause -> int array -> int -> float` qui prend en argument une clause, une valuation partielle μ_i et l'entier i et calcule et renvoie $\mathbb{P}(C_j \mid \mu_i)$.
5. En déduire une fonction `esperance_condi : fnc -> int array -> int -> float` qui prend en argument une formule, une valuation partielle μ_i et l'entier i et calcule et renvoie $\mathbb{E}(J \mid \mu_i)$.
6. En déduire une fonction `maxsat_2approx : fnc -> int array` correspondant à la 2-approximation décrite précédemment.
7. Quelle est la complexité de cette fonction ?

Palier 3

Exercice 5

On cherche à mettre en œuvre un algorithme *Branch and Bound* pour résoudre de manière exacte le problème `MAXSAT` plus efficacement qu'avec la méthode naïve. On considère pour cela une exploration de l'arbre des possibilités, correspondant à un arbre binaire de hauteur $|\mathcal{V}|$, chaque nœud x_i ayant pour enfants les affectations $\mu(x_i) = 0$ ou $\mu(x_i) = 1$.

Pour l'heuristique d'évaluation, on fait le constat suivant :

- une clause dont tous les littéraux sont déjà affectés à faux ne peut pas être satisfaite ;
- si deux clauses C_j et $C_{j'}$ non satisfaites par $\{x_0, \ldots, x_{i-1}\}$ possèdent chacune un seul littéral indéterminé, l'une x_i et l'autre $\overline{x_i}$, alors elles ne peuvent pas être simultanément satisfaites.

1. Écrire une fonction `heuristique : fnc -> int array -> int -> int` qui prend en argument une formule, une valuation partielle μ_i et l'entier i et renvoie une borne supérieure du nombre de clauses qui peuvent être satisfaites selon l'heuristique décrite précédemment.

Pour l'heuristique de branchement, on propose simplement d'affecter $\mu(x_i)$ à 1 en premier si x_i apparaît plus de fois que $\overline{x_i}$ dans des clauses non satisfaites, et 0 sinon.

2. Écrire une fonction `branchement : fnc -> int array -> int -> int` qui prend en argument une formule, une valuation partielle μ_i et l'entier i et renvoie 1 ou 0 selon que x_i apparaît plus de fois que $\overline{x_i}$ dans des clauses non satisfaites ou non.
3. En déduire une fonction `maxsat_bnb : fnc -> int array` qui détermine une valuation optimale selon un algorithme *Branch and Bound*.
4. Vérifier le résultat sur les formules $\varphi(0, 10, 20, 3)$, $\varphi(0, 15, 100, 5)$ et $\varphi(0, 20, 200, 7)$.
5. Vérifier que le nombre maximal de clauses satisfiables dans $\varphi(0, 25, 400, 8)$ est 381.
6. Calculer les ratios $\frac{J}{J^*}$ obtenus pour les FNC des deux questions précédentes, en comparant l'algorithme d'approximation et l'algorithme précédent.