La qualité de la rédaction sera prise en compte. On veillera en particulier à commenter les fonctions écrites ainsi qu'à donner leur signature (y compris pour les éventuelles fonctions auxiliaires).

Notations

- On appelle machine tout triplet (Q, Σ, δ) où Q est un ensemble fini non vide dont les éléments sont appelés *états*, Σ un ensemble fini non vide appelé *alphabet* dont les éléments sont appelés *lettres* et δ une application de $Q \times \Sigma$ dans Q appelée *fonction de transition*.
  Une machine correspond donc à un automate déterministe complet sans notion d'état initial ou d'états finaux.
- Pour un état q et une lettre x, on note $q.x = \delta(q, x)$.
- L'ensemble des mots (c'est-à-dire des concaténations de lettres) sur l'alphabet Σ est noté Σ^*.
- Le mot vide est noté ε.
- On note ux le mot obtenu par la concaténation du mot u et de la lettre x.
- On note δ^* l'extension à $Q \times \Sigma^*$ de la fonction de transition δ définie par

$$\begin{cases} \forall q \in Q,\ \delta^*(q, \varepsilon) = q \\ \forall (q, x, u) \in Q \times \Sigma \times \Sigma^*,\ \delta^*(q, xu) = \delta^*(\delta(q, x), u) \end{cases}$$

- Pour un état q de Q et un mot m de Σ^*, on note encore $q.m$ pour désigner $\delta * (q, m)$.

Pour deux états q et q', q' est dit accessible depuis q s'il existe un mot u tel que $q' = q.u$.
On dit qu'un mot m de Σ^* est *synchronisant* pour une machine (Q, Σ, δ) s'il existe un état q_0 de Q tel que pour tout état q de Q, $q.m = q_0$.
L'existence de tels mots dans certaines machines est utile car elle permet de ramener une machine dans un état particulier connu en lisant un mot donné (donc en pratique de la "réinitialiser" par une succession précise d'ordres passés à la machine réelle).
La partie I de ce problème étudie quelques considérations générales sur les mots synchronisants, la partie II est consacrée à des problèmes algorithmiques classiques, la partie III relie le problème de la satisfiabilité d'une formule logique à celui de la recherche d'un mot synchronisant de longueur donnée dans une certaine machine et enfin la partie IV s'intéresse à l'étude de l'existence d'un mot synchronisant pour une machine donnée. Les parties I, II et III peuvent être traitées indépendamment. La partie IV, plus technique, utilise la partie II.
Dans les exemples concrets de machines donnés plus loin, l'ensemble d'états peut être quelconque, de même que l'alphabet ($\Sigma = \{0, 1\}, \{a, b, c\}, \dots$). Par contre, pour la modélisation en `Caml`, l'alphabet Σ sera toujours considéré comme étant un intervalle d'entiers $[\![0, p-1]\!]$ où $p = |\Sigma|$. Une lettre correspondra donc à un entier entre 0 et $p - 1$. Un mot de Σ^* sera représenté par une liste de lettres (donc d'entiers).

```
type lettre = int;;
type mot = lettre list;;
```

De même, en Caml, l'ensemble d'états Q d'une machine sera toujours considéré comme étant l'intervalle d'entiers $[\![0, n-1]\!]$ où $n = |Q|$.

```
type etat = int;;
```

Ainsi, la fonction de transition δ d'une machine sera modélisée par une fonction Caml de signature `etat` $\rightarrow$ `lettre` $\rightarrow$ `etat`. On introduit alors le type machine

```
type machine = {n_etats : int ; n_lettres : int ; delta : etat -> lettre -> etat};;
```

où `n_etats` correspond au cardinal de Q, `n_lettres` à celui de Σ et `delta` à la fonction de transition. Pour une machine nommée `M`, les syntaxes `M.n_etats`, `M.n_lettres` ou `M.delta` permettent d'accéder à ses différents paramètres. Dans le problème, on suppose que `M.delta` s'exécute toujours en temps constant.
Par exemple, on peut créer une machine `M0` à trois états sur un alphabet à deux lettres ayant comme fonction de transition la fonction `f0` donnée ci-après.

```
let f0 etat lettre = match etat,lettre with
   |0,0 -> 1
   |0,1 -> 1
   |1,0 -> 0
   |1,1 -> 2
   |2,0 -> 0
   |2,1 -> 2;;

f0 : int -> int -> int = <fun>

let M0 = {n_etats=3;n_lettres=2;delta=f0};;
```

La figure 1 fournit une représentation de la machine M_0

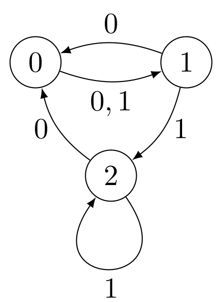

Figure 1 : La machine M_0

On pourra observer que les mots 11 et 10 sont tous les deux synchronisants pour la machine M_0.
Dans tout le sujet, si une question demande la complexité d'un programme ou d'un algorithme, on attend une complexité temporelle exprimée en $O(\dots)$.

Les programmes devront êtres écrits en Ocaml mais, par commodité, on autorisera les variables à avoir un nom commençant par une majuscule.

Partie I : Considérations générales

1. Que dire des mots synchronisants pour une machine ayant un seul état ?

Dans toute la suite du problème, on supposera que les machines ont au moins deux états.

2. On considère la machine M_1 représentée figure 2. Donner un mot synchronisant pour M_1 s'il en existe un. Justifier la réponse.

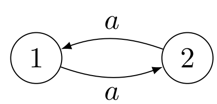

Figure 2 : La machine M_1

3. On considère la machine M_2 représentée figure 3. Donner un mot synchronisant de trois lettres pour M_2. On ne demande pas de justifier sa réponse.

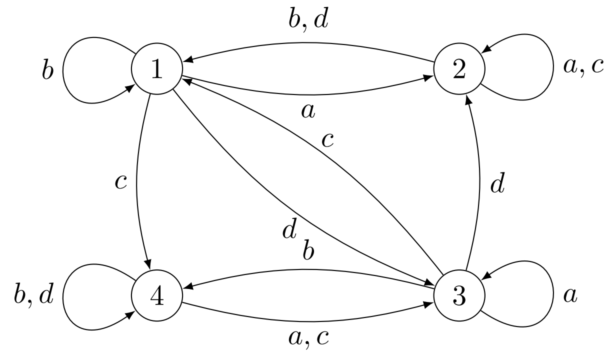

Figure 3 : La machine M_2 à 4 états

4. Ecrire une fonction `delta_etoile` de signature `machine` $\to$ `etat` $\to$ `mot` $\to$ `etat` qui, prenant en entrée une machine M, un état q et un mot u, renvoie l'état atteint par la machine M en partant de l'état q et en lisant le mot u.
5. Ecrire une fonction `est_synchronisant` de signature `machine` $\to$ `mot` $\to$ `bool` qui, prenant en entrée une machine M et un mot u, dit si le mot est synchronisant pour M.
6. Montrer que pour qu'une machine ait un mot synchronisant, il faut qu'il existe une lettre x et deux états distincts de Q, q et q', tels que $q.x = q'.x$.
7. Soit $LS(M)$ le langage des mots synchronisants d'une machine $M = (Q, \Sigma, \delta)$. On introduit la machine des parties $\widehat{M} = (\widehat{Q}, \Sigma, \widehat{\delta})$ où $\widehat{Q}$ est l'ensemble des parties de Q et où $\widehat{\delta}$ est définie par

$$\forall P \subset Q,\ \forall x \in \Sigma,\ \widehat{\delta}(P, x) = \{\delta(p, x),\ p \in P\}$$

   (a) Justifier que l'existence d'un mot synchronisant pour M se ramène à un problème d'accessibilité de certain(s) état(s) depuis certain(s) état(s) dans la machine des parties.

   (b) En déduire que le langage $LS(M)$ des mots synchronisants de la machine M est reconnaissable c'est-à-dire qu'il existe un automate fini déterministe qui reconnait le langage $LS(M)$.

   (c) Déterminer la machine des parties associée à la machine M_0 puis donner une expression régulière du langage $LS(M_0)$.
8. Montrer que si l'on sait résoudre le problème de l'existence d'un mot synchronisant, on sait dire, pour une machine M et un état q_0 de M choisi, s'il existe un mot u tel que pour tout état q de Q, le chemin menant de q à $q.u$ passe forcément par q_0.

Partie II : Algorithmes classiques

On appellera *graphe d'automate* tout couple (S, A) où S est un ensemble dont les éléments sont appelés *sommets* et A une partie de $S \times \Sigma \times S$ dont les éléments sont appelés *arcs*. Pour un arc (q, x, q'), x est *l'étiquette* de l'arc, q son *origine* et q' son *extrémité*. Un graphe d'automate correspond donc à un automate non déterministe sans notion d'état initial ou final.
Par exemple, avec

$$\begin{aligned}
\Sigma &= \{a, b\} \\
S_0 &= \{0, 1, 2, 3, 4, 5\} \\
a_0 &= \{(0, b, 0), (0, a, 3), (0, b, 2), (0, a, 1), (1, a, 1), (1, a, 2), (2, b, 1), \\
&\quad\ (2, b, 3), (2, b, 4), (3, a, 2), (4, a, 1), (4, b, 5), (5, a, 1)\}
\end{aligned}$$

le graphe d'automate $G_0 = (S_0, A_0)$ est représenté en figure 4.

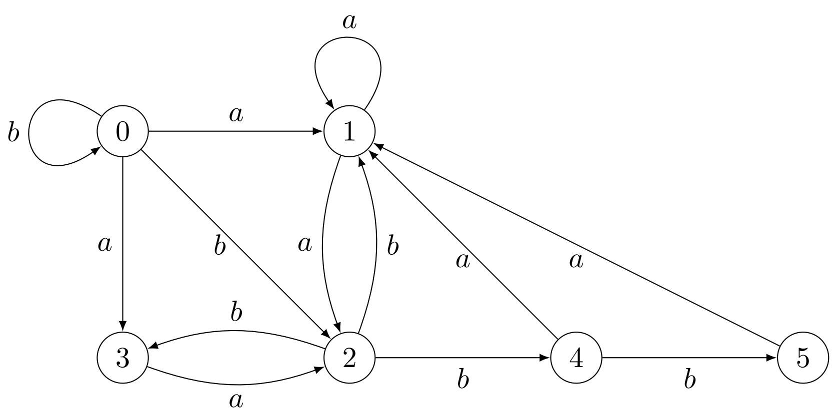

Figure 4 : Le graphe d'automate G_0

Soient s et s' deux sommets d'un graphe (S, A). On appelle chemin de s vers s' de longueur ℓ toute suite d'arcs $(s_1, x_1, s'_1), (s_2, x_2, s'_2), \ldots, (s_\ell, x_\ell, s'_\ell)$ de A telle que $s_1 = s$, $s'_\ell = s'$ et pour tout i de $[\![1, \ell - 1]\!]$, $s'_i = s_{i+1}$. L'étiquette de ce chemin est alors le mot $x_1 x_2 \ldots x_\ell$ et on dit que s' est accessible depuis s. En particulier, pour tout $s \in S$, s est accessible depuis s par le chemin vide d'étiquette ε.

Dans les programmes à écrire, un graphe aura toujours pour ensemble de sommets un intervalle d'entiers $[\![0, n-1]\!]$ et l'ensemble des arcs étiquetés par Σ (comme précédemment supposé être un intervalle $[\![0, p-1]\!]$) sera codé par un tableau de listes d'adjacences `t` : pour tout `s` $\in S$, `t.(s)` est la liste (dans n'importe quel ordre) de tous les couples (s', x) tels que (s, x, s') soit un arc du graphe. Pour des raisons de compatibilité ultérieure, les sommets (qui sont, rappelons-le, des entiers) seront codés par le type `etat`.
Ainsi, avec l'alphabet $\Sigma = \{a, b\}$, la lettre a est codée 0 et la lettre b est codée 1 ; l'ensemble des arcs du graphe G_0, dont chaque sommet est codé par son numéro, admet pour représentation `Caml` :

```
t0 : (etat*lettre) list array =[|
   [(0,1);(3,0);(2,1);(1,0)];
   [(1,0);(2,0)];
   [(1,1);(3,1);(4,1)];
   [(2,0)];
   [(1,0);(5,1)];
   [(1,0)] |]
```

9. On veut implémenter une file d'attente à l'aide d'un tableau circulaire. On définit pour cela un type particulier nommé `file` par

```
type 'a file ={
      tab:'a array;
      mutable deb:int;
      mutable fin:int;
      mutable vide:bool };;
```

   où `deb` indique l'indice du premier élément dans la file, `fin` l'indice qui suit celui du dernier élément de la file, `vide` indiquant si la file est vide. Les éléments sont rangés depuis la case `deb` jusqu'à la case précédent `fin` en repartant à la case 0 quand on arrive au bout du tableau (cf exemple). Ainsi, on peut très bien avoir l'indice `fin` plus petit que l'indice `deb`. Par exemple, la file figure 5 contient les éléments $4, 0, 1, 12$ et 8 dans cet ordre, avec `fin`$= 2$ et `deb`$= 9$.

		fin ↓							deb ↓		
12	8	7	2	5	3	1	16	3	4	0	1

Figure 5 : Un exemple de file où `fin`<`deb`

On rappelle qu'un champ mutable peut voir sa valeur modifiée.
Par exemple, la syntaxe `f.deb ← 0` affecte la valeur 0 au champ `deb` de la file `f`.

(a) Ecrire une fonction `ajoute` de signature `'a file → 'a → unit` telle que `ajoute f x` ajoute `x` à la fin de la file d'attente `f`. Si c'est impossible, la fonction devra renvoyer un message d'erreur, en utilisant l'instruction `failwith "File pleine"`.

(b) Ecrire une fonction `retire` de signature `'a file → 'a` telle que `retire f` retire l'élément en tête de la file d'attente et la renvoie. Si c'est impossible, la fonction devra renvoyer un message d'erreur.

(c) Quelle est la complexité de ces fonctions ?

On considère l'algorithme 1 s'appliquant à un graphe d'automates $G = (S, A)$ et à un ensemble de sommets E (on note $n = |S|$ et ∞, *vide* et *rien* des valeurs particulières).

```
créer une file d'attente F, vide au départ
créer un tableau D dont les cases sont indexées par S et initialisées à ∞
créer un tableau P dont les cases sont indexées par S et initialisées à vide
créer une variable c initialisée à n
pour tout s ∈ E faire
   insérer s à la fin de la file d'attente F
   fixer D[s] à 0
   fixer P[s] à rien
   diminuer c de 1
fin pour
tant que F n'est pas vide faire
   extraire le sommet s qui est en tête de F
   pour tout arc (s, y, s') ∈ A tel que D[s'] = ∞ faire
      fixer D[s'] à D[s] + 1
      fixer P[s'] à (s, y)
      insérer s' à la fin de la file d'attente F
      diminuer c de 1
   fin pour
fin tant que
renvoyer (c, D, P)
```

Algorithme 1

10. Justifier que l'algorithme 1 termine toujours.
11. Donner la complexité de cet algorithme en fonction de $|S|$ et $|A|$. On justifiera la réponse.
12. Justifier qu'au début de chaque passage dans la boucle "**tant que** F n'est pas vide", si F contient dans l'ordre les sommets $s_1, s_2, \ldots, s_r$, alors $D[s_1] \leq D[s_2] \leq \cdots \leq D[s_r]$ et $D[s_r] - D[s_1] \leq 1$.
13. Pour s sommet de G, on note d_s la distance de E à s c'est à dire la longueur d'un plus court chemin d'un sommet de E à s (avec la convention $d_s = \infty$ s'il n'existe pas de tel chemin).
    (a) Justifier brièvement qu'à la fin de l'algorithme, pour tout sommet s, $D[s] \neq \infty$ si et seulement si s est accessible depuis un sommet de E et que $d_s \leq D[s]$. Que désigne alors c ?
    (b) Montrer qu'en fait, à la fin, on a pour tout sommet s, $D[s] = d_s$. Que vaut alors $P[s]$?
14. Ecrire une fonction

```
accessibles : ((etat*lettre) list) array -> etat list
                                          -> int*int array*(etat*lettre) array
```

prenant en entrée un graphe d'automate (sous forme de son tableau de listes d'adjacence `t`) et un ensemble `E` de sommets (sous forme d'une liste d'états) et qui renvoie le triplet (c, D, P) calculé selon l'algorithme précédent. Les constantes ∞, *vide* et *rien* seront respectivement codées par -1, $(-2, -1)$ et $(-1, -1)$ dans cette fonction.

15. Ecrire une fonction `chemin` de signature `etat` → `(etat*lettre) array` → `mot` qui, prenant en entrée un sommet `s` et le tableau `P` calculé à l'aide de la fonction `accessibles` sur un graphe G et un ensemble E, renvoie un mot de longueur minimale qui est l'étiquette d'un chemin d'un sommet de E à s (ou un message d'erreur s'il n'en existe pas).

Partie III : Réduction SAT

On s'intéresse dans cette partie à la satisfiabilité d'une formule logique portant sur des variables propositionnelles $x_1, \ldots, x_m$. On note classiquement $\wedge$ le connecteur logique “et”, $\vee$ le connecteur “ou” et $\overline{f}$ la négation d'une formule f.
On appelle littéral une formule constituée d'une variable x_i ou de sa négation $\overline{x_i}$, on appelle clause une disjonction de littéraux.
Considérons une formule logique sous forme normale conjonctive, c'est à dire sous la forme d'une conjonction de clauses. Par exemple,

$$F_1 = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_4}) \wedge (x_2 \vee \overline{x_3} \vee x_4)$$

est une formule sous forme normale conjonctive formée de trois clauses et portant sur quatre variables propositionnelles x_1, x_2, x_3 et x_4.
Soit F une formule sous forme normale conjonctive, composée de n clauses et faisant intervenir m variables. On suppose les clauses numérotées $c_1, c_2, \ldots, c_n$. On veut ramener le problème de la satisfiabilité d'une telle formule au problème de la recherche d'un mot synchronisant de longueur $\leq m$ sur une certaine machine. On introduit pour cela la machine suivante associée à F :

- Q est formé de $mn+n+1$ états, un état particulier noté f et $n(m+1)$ autres états qu'on notera $q_{i,j}$ avec $(i,j) \in [\![1,n]\!] \times [\![1,m+1]\!]$;
- $\Sigma = \{0,1\}$:
- δ est défini par
  - f est un état puits, c'est à dire $\delta(f,0) = \delta(f,1) = f$,
  - pour tout entier $i \in [\![1,n]\!]$, $\delta(q_{i,m+1},0) = \delta(q_{i,m+1},1) = f$,
  - pour tous $i \in [\![1,n]\!]$ et $j \in [\![1,m]\!]$,

$$\delta(q_{i,j},1) = \begin{cases} f & \text{si le littéral } x_j \text{ apparaît dans la clause } c_i \\ q_{i,j+1} & \text{sinon} \end{cases}$$

$$\delta(q_{i,j},0) = \begin{cases} f & \text{si le littéral } \overline{x_j} \text{ apparaît dans la clause } c_i \\ q_{i,j+1} & \text{sinon} \end{cases}$$

16. Représenter la machine associée à la formule F_1.
17. Donner une distribution de vérité $(v_1, v_2, v_3, v_4) \in [\![0,1]\!]^4$ (la valeur v_i étant associée à la variable x_i) satisfaisant F_1. Le mot $v_1v_2v_3v_4$ est-il synchronisant.
18. Montrer que tout mot u de longueur $m+1$ est synchronisant. À quelle condition sur les $q_{i,1}.u$ un mot de longueur m est-il synchronisant ?
19. Montrer que si la formule F est satisfiable, toute distribution de vérité la satisfaisant donne un mot synchronisant de longueur m pour l'automate.
20. Inversement, prouver que si l'automate dispose d'un mot synchronisant de longueur $\leq m$, F est satisfiable. Donner alors une distribution de vérité convenable.

Partie IV : Existence

On reprend dans cette partie le problème de l'existence d'un mot synchronisant pour une machine M.

21. Soit $M = (Q, \Sigma, \delta)$ une machine.
    Pour toute partie E de Q et tout mot u de Σ^*, on note $E.u = \{q.u,\ q \in E\}$.
    (a) Soit u un mot synchronisant de M et $u_0, u_1, \ldots, u_r$ une suite de préfixes de u rangés dans l'ordre croissant de leur longueur et telle que $u_r = u$. Que peut-on dire de la suite des cardinaux $|Q.u_i|$?

(b) Montrer qu'il existe un mot synchronisant si et seulement s'il existe pour tout couple d'états (q, q') de Q^2 un mot $u_{q,q'}$ tel que $q.u_{q,q'} = q'.u_{q,q'}$.

On veut se servir du critère établi ci-dessus pour déterminer s'il existe un mot synchronisant. Pour cela, on associe à la machine M la machine $\widetilde{M} = (\widetilde{Q}, \Sigma, \widetilde{\delta})$ définie par :

- $\widetilde{Q}$ est formé des parties à un ou deux éléments de Q ;
- $\widetilde{\delta}$ est définie par $\forall (E, x) \in \widetilde{Q} \times \Sigma,\ \widetilde{\delta}(E) = \{\delta(q, x),\ q \in E\}$.

22. Si $n = |Q|$, que vaut $\widetilde{n} = |\widetilde{Q}|$?

On a dit que pour la modélisation informatique, l'ensemble d'états d'une machine doit être modélisée par un intervalle $[\![0, n-1]\!]$. $\widetilde{Q}$ doit donc être modélisé par l'intervalle $[\![0, \widetilde{n}-1]\!]$. Soit φ_n une bijection de $\widetilde{Q}$ sur $[\![0, \widetilde{n}-1]\!]$. On suppose qu'on dispose d'une fonction `set_to_nb` de signature `int` $\to$ `(etat list)` $\to$ `etat` telle que `set_to_nb n l` pour `n` représentant un élément $n \in \mathbb{N}^*$ et `l` représentant une liste ℓ d'états renvoie

$$\begin{cases} \varphi_n(\{i\}) & \text{si } \ \ell = [i],\ 0 \leq i \leq n-1 \\ \varphi_n(\{i, j\}) & \text{si } \ \ell = [i; j],\ 0 \leq i < j \leq n-1 \end{cases}$$

On suppose qu'on dispose aussi d'une fonction réciproque `nb_to_set` de signature `int` $\to$ `etat` $\to$ `(etat list)` telle que `nb_to_set n q` pour $n \in \mathbb{N}^*$ et $q \in [\![0, \widetilde{n}-1]\!]$ renvoie une liste d'états de la forme $[i]$ ou $[i; j]$ (avec $i < j$) correspondant à $\varphi_n^{-1}(q)$. Ces deux fonctions de conversion sont supposées agir en temps constant.

Enfin, pour ne pas confondre un état de $\widetilde{Q}$ avec sa représentation informatique par un entier, on notera $\overline{q}$ l'entier associé à l'état q.

23. Ecrire une fonction `delta2` de signature `machine` $\to$ `etat` $\to$ `lettre` $\to$ `etat` qui prenant en entrée une machine M, un état $\overline{q}$ de $\widetilde{Q}$ et une lettre x, renvoie l'état de $\widetilde{Q}$ atteint en lisant la lettre x depuis l'état q dans $\widetilde{M}$.

24. Il est clair qu'à la machine $\widetilde{M}$, on peut associer un graphe d'automate $\widetilde{G}$ dont l'ensemble des sommets est $\widetilde{Q}$ et dont l'ensemble des arcs est $\{(q, x, \widetilde{\delta}(q, x)),\ (q, x) \in \widetilde{Q} \times \Sigma\}$. On associe alors à $\widetilde{G}$ le graphe retourné $\widetilde{G}_R$ qui a les mêmes sommets que $\widetilde{G}$ mais dont les arcs sont retournés (i.e. (q, x, q') est un arc de $\widetilde{G}_R$ si et seulement si (q', x, q) est un arc de $\widetilde{G}$).
Ecrire une fonction `retourne_machine` de signature `machine` $\to$ `((etat*lettre) list) array` qui à partir d'une machine M, calculer le tableau V des listes d'adjacence du graphe $\widetilde{G}_R$.

25. Justifier qu'il suffit d'appliquer la fonction `accessibles` de la partie **2** au graphe $\widetilde{G}_R$ et à l'ensemble des sommets de $\widetilde{G}_R$ correspondant à des singletons pour déterminer si la machine M possède un mot synchronisant.

26. Ecrire une fonction `existe_synchronisant` de signature `machine` $\to$ `bool` qui dit si une machine possède un mot synchronisant.

Jan Černỳ, chercheur slovaque, a conjecturé au milieu des années 60 que si une machine à n états possédait un mot synchronisant, elle en avait un de longueur $\leq (n-1)^2$. La construction faite dans la partie 3 affirme que la recherche, dans une machine, d'un mot synchronisant de longueur $\leq m$ fixé est au moins aussi difficile en terme de complexité que celui de la satisfiabilité d'une formule logique à m variables sous forme normale conjonctive (qu'on sait être un problème "difficile").