université de BORDEAUX

Collège Sciences et technologies

Année	2022-2023	Type	Devoir Surveillé
Master	Informatique		
Code UE	4TIN705U	Épreuve	Systèmes d'Exploitation
Date	18/11/2022	Documents	Non autorisés
Début	10h30	Durée	1h30

La plupart des questions peuvent être traitées même si vous n'avez pas répondu aux précédentes. *À la fin du sujet, vous trouverez un memento vous rappelant la syntaxe de quelques fonctions utiles.*

1 Questions de cours (échauffement)

Question 1 Définissez brièvement ce que l'on appelle "ordonnanceur" dans un système d'exploitation. Quand et par qui le code d'ordonnancement est-il exécuté (listez des situations bien précises) ? Dans un système Unix, quels sont les principaux paramètres pris en compte pour le choix du prochain processus à exécuter ?

Question 2 Dans les noyaux Linux 2.4.x, l'ordonnancement des processus s'appuie sur un système de *crédits* qui sont débités à l'issue de chaque quantum de temps utilisé. Est-ce que ce système garantit une alternance d'exécution entre deux processus de même priorité qui utilisent tous deux le processeur de manière intensive (i.e. sans jamais effectuer d'appel système) ? Expliquez.

2 Synchronisation

On souhaite pouvoir lancer des traitements en « tâches de fond » dans un programme, c'est-à-dire déléguer à un ou plusieurs *threads* l'exécution de fonctions pendant que le programme principal continue son exécution. On appelle *tâche* un couple (fonction, argument) dont l'exécution est ainsi effectuée de manière asynchrone.

Voici un exemple de programme que l'on souhaiterait pouvoir exécuter :

```
1  void f (void *arg)
2  { ... }
3
4  void g (void *arg)
5  { ... }
6
7  int main ()
8  {
9    tasks_init ();
10
11   tasks_submit (f, "task 1");
12   tasks_submit (g, "task 2");
13   tasks_submit (f, "task 3");
14
15   // Do some work while tasks are running in background...
16 }
```

L'appel `tasks_init` (ligne 9) initialise une file de tâches d'une capacité finie, et crée un pool de threads prêts à extraire des tâches de la file dès qu'il y en aura. Chaque appel à `tasks_submit` (lignes 11–13) dépose une nouvelle tâche dans la file. L'opération est bloquante si la file est pleine.

Si le pool se limite à un seul thread, alors les tâches seront exécutées les unes après les autres. Dans le cas d'un pool de plusieurs threads, certaines tâches pourront être exécutées en parallèle (dans la limite du nombre de threads disponibles).

Une implémentation préliminaire vous est fournie ci-dessous. Pour l'instant, on suppose que le programme principal se termine systématiquement (miraculeusement même) après que toutes les tâches aient été exécutées.

```c
typedef struct {
  func_t f;
  void *arg;
} task_t;

// Implementation details of queue are omitted
#define MAX_QUEUE ...
int __push (const task_t *t) { ... }
int __pop  (task_t *t) { ... }
unsigned __size (void) { ... }

int push (const task_t *t)
{
  return __push (t);
}

int pop (task_t *t)
{
  return __pop (t);
}
```

```c
void pop_and_execute (void)
{
  task_t t;

  if (pop (&t) == 0)
    t.f (t.arg);
}

void thread_body (void)
{
  // each thread executes an infinite loop
  for (;;)
    pop_and_execute ();
}

void tasks_submit (func_t f, void *arg)
{
  task_t t = { f, arg };
  while (push (&t) == -1)
    /* nothing */ ;
}
```

Cette implémentation s'appuie sur une file de tâches accessible au travers des primitives `__push`, `__pop` et `__size`. L'implémentation importe peu [1], on sait juste que la file a une capacité maximale égale à `MAX_QUEUE`, et qu'aucune précaution particulière n'a été prise pour que ces primitives fonctionnent lorsqu'elles sont appelées de manière concurrente. Si la file est pleine, `__push` échoue et renvoie -1. De même, si la file est vide, `__pop` échoue et renvoie -1.

On ne s'intéressera pas au code qui crée les threads dans `task_init`. On sait juste que tous les threads créés exécutent la fonction `thread_body`.

Question 1 En l'absence de garantie sur l'implémentation des fonctions `__push`, `__pop` et `__size`, a-t-on tout de même l'assurance que le programme fonctionne correctement lorsqu'un seul thread est créé par `task_init` ? Expliquez. Par ailleurs, le code risque-t-il de solliciter maladroitement les processeurs de la machine ? Pourquoi ?

Question 2 En utilisant les outils associés aux moniteurs de Hoare (cf memento en fin de sujet), ajoutez la synchronisation nécessaire aux primitives `push` et `pop` de sorte que

— `task_submit` soit bloquante lorsque la file est pleine ;

— `pop_and_execute` soit bloquante lorsque la file est vide.

Indiquez quelles sont les variables globales ajoutées, et précisez à quel moment elles doivent être initialisées.

Question 3 On souhaite maintenant disposer d'une fonction **void** `tasks_wait_all` (**void**) qui sera appelée par le programme principal pour attendre que toutes les tâches soumises soient exécutées. Donnez le code de cette fonction, et indiquez quelles sont les modifications éventuelles à apporter aux autres fonctions. Attention à bien attendre qu'il n'y ait plus de tâche en cours (il ne s'agit pas seulement d'attendre qu'il n'y ait plus rien dans la file).

Question 4 On décide maintenant de distinguer deux types de tâches : les tâches *normales*, et les tâches *exclusives*. Ces dernières sont uniquement exclusives entre elles : il ne doit jamais y avoir deux tâches exclusives exécutées en même temps. L'exécution des tâches normales n'obéit à aucune contrainte de ce type.

Pour ce faire, on ajoute un paramètre supplémentaire à `task_submit` ainsi qu'un champ à la structure `task_t` :

```c
typedef enum { REGULAR, EXCLUSIVE } task_type_t;

typedef struct {
  func_t f;
  void *arg;
  task_type_t type;
} task_t;

void tasks_submit (func_t f, void *arg, task_type_t type);
```

1. On n'essaiera pas de modifier ces fonctions.

Voici un exemple de programme principal :

```
int main ()
{
  tasks_init (nbthreads);

  tasks_submit (f, "task 1", EXCLUSIVE);
  tasks_submit (f, "task 2", EXCLUSIVE);
  tasks_submit (f, "task 3", REGULAR);

  tasks_wait_all ();
}
```

En supposant une file de capacité au moins égale à 3, voici quelques scenarii d'exécution :

— Avec un pool d'un seul thread, les 3 tâches sont exécutées séquentiellement ;

— Avec un pool de deux threads, la tâche 1 est exécutée d'abord, et une fois terminée les tâches 2 et 3 peuvent être exécutées en parallèle ;

— Avec un pool de trois threads ou plus, la tâche 1 et la tâche 3 peuvent s'exécuter en parallèle, et une fois la tâche 1 terminée, la tâche 2 pourra démarrer.

Proposez une solution simple et indiquez ce qu'il faut modifier dans les fonctions du code pour assurer la synchronisation demandée. Attention, la synchronisation ajoutée ne doit pas empêcher l'accès à la file pendant qu'une tâche exclusive est en cours...

Question 5 On souhaite mettre en place une solution plus sophistiquée au problème posé à la question précédente. En effet, dans l'exemple présenté en question 4, et dans le cas où le pool ne serait constitué que de deux threads, on voudrait que les tâches 1 et 3 s'exécutent en parallèle dès le début.

Autrement dit, le thread qui récupère la tâche 2 devrait pouvoir s'apercevoir qu'il y a déjà une tâche exclusive en cours et qu'il est dommage d'attendre alors qu'il existe au moins une tâche normale plus loin dans la file. L'idée serait alors que le thread remette dans la file les tâches exclusives qu'il récupère jusqu'à ce qu'il récupère une tâche normale.

Donnez le code de cette dernière version.

Memento

```
typedef ... mutex_t;
void mutex_lock(mutex_t *m);
void mutex_unlock(mutex_t *m);
```

```
typedef ... cond_t;
void cond_wait(cond_t *c, mutex_t *m);
void cond_signal(cond_t *c);
void cond_bcast(cond_t *c);
```

Master CSI 1 — 2022-2023

Arithmétique : Examen du 14 décembre 2022

Master Sciences et Technologies, mention Mathématiques ou Informatique, parcours Cryptologie et Sécurité informatique

Responsable : Gilles Zémor

Durée : 3h. Sans document. Les exercices sont indépendants.

– EXERCICE 1. Soit A l'anneau $\mathbb{F}_2[X]/(X^5+X)$. Combien d'éléments contient le groupe A^* des éléments inversibles de A ? Montrer que tout élément x de A^* est tel que $x^4 = 1$. Le groupe A^* est-il cyclique ?

– EXERCICE 2.

a) Quels sont les degrés des facteurs irréductibles de $X^{27}+1$ dans $\mathbb{F}_2[X]$?

b) En écrivant que $X^{27} = (X^9)^3$, et en utilisant la décomposition de X^3+1 en facteurs irréductibles dans $\mathbb{F}_2[X]$, trouver la décomposition en facteurs irréductibles de $X^{27}+1$.

c) Combien l'anneau $A = \mathbb{Z}/81\mathbb{Z}$ a-t-il d'éléments inversibles ?

d) Calculer 2^{18} et 2^{27} dans A et en déduire que 2 est un générateur du groupe multiplicatif des éléments inversibles de A.

e) En déduire le plus gros degré des facteurs irréductibles de $X^{81}+1$.

f) En écrivant $X^{81} = (X^{27})^3$, en déduire la décomposition en facteurs irréductibles de $X^{81}+1$ sur $\mathbb{F}_2[X]$.

– EXERCICE 3. On considère les suites binaires $a = (a_i)_{i\geqslant 0}$ engendrées par la récurrence linéaire

$$a_i = a_{i-5} + a_{i-6} \qquad (1)$$

a) Quel est le polynôme de rétroaction $h(X)$ de cette récurrence linéaire ? Montrer qu'il est irréductible primitif.

b) Quelle est la période d'une solution non nulle de la récurrence (1) ?

c) Soit α une racine de $h(X)$ dans un corps d'extension de $\mathbb{F}_2$. Quel est le degré de l'extension $\mathbb{F}_2(\alpha)$ de $\mathbb{F}_2$? En notant Tr() l'application trace de $\mathbb{F}_2(\alpha)$ dans $\mathbb{F}_2$, calculer $\mathrm{Tr}(\alpha)$.

d) Utiliser une propriété simple de la fonction trace pour montrer que dans le cas présent, $\mathrm{Tr}(\alpha^3) = \mathrm{Tr}(\alpha)$.

e) En déduire, sans faire aucun calcul de trace supplémentaire, les 10 premiers symboles de la suite binaire $\mathrm{Tr}(\alpha^i)$, $i = 0, 1, \ldots, 9$.

f) Soit $\beta = \alpha^7$. Calculer β^3 et β^6 dans la base $1, \alpha, \ldots, \alpha^5$ et en déduire le polynôme minimal de β.

g) Quelle est la période de la suite $\mathrm{Tr}(\beta^i)$, quelle récurrence linéaire suit-elle, et quel est son polynôme de rétroaction ?

h) Donner les valeurs de $\mathrm{Tr}(\beta^i)$, $i = 0, 1, \ldots$, sur une période. Évitez de vous épuiser à faire de nouveaux calculs de trace. Pensez à utiliser les propriétés de la trace et la forme du polynôme minimal de β trouvé en f).

– EXERCICE 4. Étudier les puissances de X de la forme X^{2^i} modulo $X^{15} + X + 1$ et en déduire que $X^{15} + X + 1$ est irréductible. Il sera utile de montrer que si un polynôme de degré 15 est un produit de polynômes de degrés 3 ou 5 alors il est soit un produit de polynômes de degré 3, soit un produit de polynômes de degré 5.

– EXERCICE 5. Soit C le code cyclique de longueur 15 de polynôme générateur $g(X) = X^4 + X + 1$. Soit le mot $\mathbf{y}$ de $\mathbb{F}_2^{15}$ représenté par le polynôme $\mathbf{y}(X) = 1 + X + X^3 + X^7 + X^9$. Trouver le mot $\mathbf{c}$ du code C à distance de Hamming 1 de $\mathbf{y}$.

– EXERCICE 6. Soit $g(X) = 1 + X^2 + X^4 + X^5 = (1 + X)(1 + X + X^4) \in \mathbb{F}_2[X]$. Soit C le code cyclique de longueur 15 de polynôme générateur $g(X)$. On rappelle qu'on adopte la convention que chaque n-uple $(c_0, c_1, \ldots c_{14})$ est représenté par le polynôme $c_0 + c_1 X + \cdots c_{14} X^{14}$.

a) Quelle est la distance minimale de C ?

b) Par souci de lisibilité, on écrit les mots de longueur 15 sous la forme abc où a, b, c sont chacun des quintuplets binaires (éléments de $\mathbb{F}_2^5$). Montrer que pour chaque valeur de ab, il existe une unique valeur de c telle que abc soit un mot du code C.

c) Soient $a = 00000$ et $b = 00001$. Trouver la valeur de c pour laquelle abc est un mot du code C. On pourra remarquer que abc est dans le code C si et seulement si cab l'est, et calculer un reste modulo $g(X)$.

d) Le polynôme $c(X) = 1 + X + X^3 + c_7 X^7 + c_{12} X^{12} + X^{13} + c_{14} X^{14}$ représente un mot du code C, mais on ne connaît pas les valeurs de c_7, c_{12} et c_{14}. Pour les trouver, commencez par écrire une table donnant les valeurs de $\alpha^i, i = 0, 1 \ldots, 14$, où α est la classe de X dans $\mathbb{F}_{16} = \mathbb{F}_2[X]/(X^4 + X + 1)$. Puis, utilisez la propriété $c(\alpha) = 0$ pour déterminer entièrement $c(X)$.

e) On définit maintenant le polynôme $G(X) = g(X)(X^4 + X^3 + X^2 + X + 1)$. Soit $D \subset \mathbb{F}_2^{15}$ le code cyclique de polynôme générateur $G(X)$. Quelle est la distance minimale de D ?

f) Montrer que les mots $(c_0, c_1, \ldots, c_{14})$ du code D sont les solutions de la récurrence linéaire

$$c_i = c_{i-1} + c_{i-2} + c_{i-3} + c_{i-6}$$

où l'addition des indices s'entend modulo 15.

Master CSI 1 2022-2023

Arithmétique : DS du 9 novembre 2022

Master Sciences et Technologies, mention Mathématiques ou Informatique, parcours Cryptologie et Sécurité informatique

Responsable : Gilles Zémor

Durée : 1h30. Sans document. Les exercices sont indépendants.

– EXERCICE 1. Soit $A = \mathbb{F}_3[X]/(X^3 + X)$.

a) Combien l'anneau A contient-il d'éléments ? Combien d'éléments contient le groupe multiplicatif A^* des éléments inversibles de A ?

b) Montrer que tout élément α de A vérifie $\alpha^9 = \alpha$. En déduire que le groupe multiplicatif A^* n'est pas cyclique.

– EXERCICE 2. Tous les polynômes considérés sont dans $\mathbb{F}_2[X]$.

a) Calculer X^{2^i} modulo $X^7 + X^3 + 1$, $i = 0, 1, 2, \ldots, 7$ et en déduire que $X^7 + X^3 + 1$ est irréductible.

b) Soit $P(X) = X^8 + X^4 + X^3 + X + 1 \in \mathbb{F}_2[X]$. On admettra que le plus petit entier $i \geqslant 1$ tel que $X^{2^i} = X \bmod P(X)$ vaut 8. Expliquer comment on peut en déduire que $P(X)$ est irréductible.

– EXERCICE 3. On considère $\mathbb{F}_{16}$, le corps à 16 éléments. Soit γ un élément primitif de $\mathbb{F}_{16}$.

a) Quelles sont les puissances de γ, qui avec 0 et 1 constituent un sous-corps K de $\mathbb{F}_{16}$ isomorphe à $\mathbb{F}_4$?

b) Montrer que pour tout $\alpha \in \mathbb{F}_{16}$, $\alpha + \alpha^4 \in K$.

c) Soit $\alpha \in \mathbb{F}_{16}$, $\alpha \notin K$. Que vaut $[\mathbb{F}_{16} : K]$?

d) En déduire le degré du polynôme minimal de $\alpha \notin K$ dans $K[X]$. Donner une expression des coefficients de ce polynôme minimal en fonction de α.

– EXERCICE 4.

a) On considère le polynôme $X^6 + X^3 + 1$ dans $\mathbb{F}_2[X]$. Calculer X^{64} modulo $X^6 + X^3 + 1$ et en déduire, en faisant très attention, que $X^6 + X^3 + 1$ est irréductible.

b) $X^6 + X^3 + 1$ est-il primitif ?

c) Soit α une racine de $X^6 + X^3 + 1$ dans $\mathbb{F}_{64}$. Soit $\beta = \alpha^5 + \alpha^2 + \alpha$. Calculer $[\mathbb{F}_2(\beta) : \mathbb{F}_2]$.

d) Montrer que $\alpha + 1$ est primitif.

e) Trouver le polynôme minimal de $\alpha + 1$.

	ANNÉE UNIVERSITAIRE 2022/2023 **4TMA701U Calcul Formel** **Devoir Surveillé** **Date : 09/11/2022 Heure : 15h30 Durée : 1h30** Documents non autorisés.	**Collège Sciences et Technologies**

Vous rendrez à la fin de l'examen une copie papier ainsi qu'un fichier sage contenant vos programmes (lisible, commenté et nettoyé si possible..) au format DS-Nom-Prenom.ipynb (feuille Jupyter) ou DS-Nom-Prenom.sage (fichier texte). Le fichier est à envoyer par e-mail à votre enseignant.e de TD (christine.bachoc@u-bordeaux.fr ou gilles.zemor@u-bordeaux.fr).

Exercice 1 Soient $p_1 < p_2 < \cdots < p_m < p_{m+1} < p_{m+2}$ une suite strictement croissante de $m+2$ nombres premiers. On note M le produit des m premiers termes de la suite, soit $M = p_1 p_2 \cdots p_m$.

Dans toute la suite, pour un entier x, et pour tout $i \in [1, \ldots, m+2]$, on note x_i son reste modulo p_i.

1. Soit x un entier tel que $0 \leq x < M$. Montrez que si on vous donne m parmi les $m+2$ valeurs des x_i, alors vous pouvez reconstituer x sans ambiguïté (expliquez comment et justifiez votre réponse).

   Indication : Commencez par traiter le cas où les m valeurs données sont les m premières et pensez à utiliser un célèbre théorème d'arithmétique ..

   Exemple numérique : $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (2, 3, 5, 7, 11, 13, 17)$, avec $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (*, 0, 2, 6, 9, *, 16)$, les "*" représentant les valeurs manquantes. Que vaut x ?

   Vous pouvez utilisez la fonction **crt** *de sage ; expliquez votre algorithme et écrivez une fonction qui prend en entrée une liste de premiers et une liste de restes dans laquelle les deux restes manquent (remplacez-les par exemple par des -1), et sort x.*

2. Soit x un entier tel que $0 \leq x < M$. On vous donne maintenant les $m+2$ restes modulo p_i de x, mais une de ces valeurs, vous ne savez pas laquelle, est fausse. Dit autrement, on dispose de $(y_1, y_2, \ldots, y_{m+1}, y_{m+2})$ avec $y_i = x_i$ pour toutes les valeurs de i *sauf une.* Montrez que vous pouvez retrouver l'entier x sans ambiguïté et expliquez comment.

   Exemple numérique : les mêmes p_i que précédemment, avec $(y_1, y_2, y_3, y_4, y_5, y_6, y_7) = (1, 1, 3, 3, 5, 3, 8)$. Que vaut x ?

   Vous pouvez utilisez la fonction **crt** *de sage ; expliquez votre algorithme et écrivez une fonction qui prend en entrée une liste de premiers et une liste de restes, et sort x.*

Exercice 2 Dans cet exercice vous allez étudier une variante du test de primalité de Pocklington Lehmer vu en cours.

Dans tout l'exercice, n est un entier impair, tel que $n-1 = pu$ avec p un nombre premier impair, $p > \sqrt{n}$, et $\text{pgcd}(p,u) = 1$. On considère l'hypothèse (H) suivante :

$$\text{(H)} \quad \text{Il existe un entier } b,\ 1 \leq b < n \text{ tel que : } \begin{cases} b^{n-1} = 1 \bmod n \\ \text{pgcd}(b^u - 1, n) = 1 \end{cases}$$

1. Dans cette question vous allez montrer que si (H) est vérifiée alors n est premier. On suppose donc (H).
   a) Supposons que n a un diviseur premier q avec $q \leq \sqrt{n}$. Soit $c = b^u \bmod q$. Montrez que $c \neq 1 \bmod q$ mais que $c^p = 1 \bmod q$.
   b) En déduire que n est premier.
2. Déduire de la question 1) un test de primalité pour n, prenant en entrées n, p et b comme ci-dessus et sortant "n est premier" ou "n est composé" ou "échec, on ne peut pas conclure". Programmez ce test sous la forme d'une fonction sage.
   Indication : Attention de ne pas faire intervenir dans l'exécution des entiers plus grands que n. Vous pouvez utiliser la fonction **IntegerModRing()** .
3. Analysez l'ordre de grandeur de la complexité binaire de votre test en fonction de la taille binaire de n.
4. Construire une liste de nombres premiers de plus en plus grands en partant de $p_1 = 1000003$ et en cherchant grâce à votre test de primalité avec $b = 2$, pour $i \geq 1$, un nombre premier p_{i+1} de la forme $p_i(10^{e_i - 1} + k) + 1$, où $10^{e_i} \leq p_i < 10^{e_i+1}$ (essayer successivement $k = 2, 4, 6, \ldots$). Vous devriez pouvoir dépasser 100 chiffres décimaux.

	ANNÉE UNIVERSITAIRE 2022/2023 **4TMA701U Calcul Formel** **Examen terminal session 1** **Date : 13/12/2022 Heure : 9h Durée : 3h** Accès autorisé aux feuilles TD sur Moodle.	**Collège Sciences et Technologies**

Vous rendrez à la fin de l'examen une copie papier ainsi qu'un fichier sage contenant vos programmes (lisible, commenté et nettoyé si possible..) au format EX-Nom-Prenom.ipynb (feuille Jupyter) ou EX-Nom-Prenom.sage (fichier texte). Le fichier est à envoyer par e-mail à christine.bachoc@u-bordeaux.fr

Exercice 1 Nous avons vu en cours qu'il existe des algorithmes (utilisant FFT) de complexité algébrique $\tilde{\mathrm{O}}(n)$ pour la multiplication et pour la division euclidienne dans $K[X]$ lorsque le degré des polynômes est inférieur à n, ainsi que des algorithmes (utilisant FFT) de complexité binaire $\tilde{\mathrm{O}}(s)$ pour la multiplication et pour la division euclidienne des nombres entiers de taille binaire inférieure à s. À partir de là déterminez la complexité de :

1. La multiplication dans $R = K[X]/(P)$ où $P \in K[X]$ est de degré k (on demande la complexité algébrique, exprimée en fonction de k. Explicitez la représentation des éléments de R et les étapes nécessaires à la multiplication dans R).
2. La multiplication dans $\mathbb{F}_q$ où $q = p^k$, p premier (on demande la complexité binaire, exprimée en fonction de q, ou de p, k. On explicitera la représentation binaire des éléments de $\mathbb{F}_q$).
3. La multiplication dans $A = \mathbb{F}_q[X]/(P)$ où $P \in \mathbb{F}_q[X]$ est de degré d (on demande la complexité binaire, exprimée en fonction de q, d, ou de p, k, d).
4. L'exponentiation dans $A = \mathbb{F}_q[X]/(P)$: calcul de a^n pour $a \in A$ et $n \in \mathbb{N}$ (on demande la complexité binaire, exprimée en fonction de q, d, n, ou de p, k, d, n).

Exercice 2 Cet exercice porte sur un algorithme de partage de secret. Une personne A détient un secret $s \in \mathbb{N}$ qu'elle souhaite partager avec un groupe de n personnes $B_1, \ldots, B_n$. Toutefois elle souhaite que s reste inconnu de chacun des B_i, et même de toute partie incomplète du groupe. Pour cela elle construit à partir de s des valeurs $x_1, \ldots, x_n$, et transmet x_i à B_i $(i = 1, \ldots, n)$. La mise en commun des n valeurs x_i doit permettre au groupe de reconstruire s, mais aucune information sur s ne doit pouvoir être obtenue à partir d'un sous-ensemble strict des x_i.

A procède ainsi : elle fixe un corps fini $\mathbb{Z}/p\mathbb{Z}$, avec $p > n, s$, ainsi que $(t_1, \ldots, t_n) \in (\mathbb{Z}/p\mathbb{Z})^n$ avec $t_i \neq t_j$ pour tout $i \neq j$. Ces données sont publiques. Pour calculer les x_i elle tire au hasard $(a_1, \ldots, a_{n-1}) \in (\mathbb{Z}/p\mathbb{Z})^{n-1}$ et pose $P = s + \sum_{k=1}^{n-1} a_k x^k \in \mathbb{Z}/p\mathbb{Z}[x]$. Puis elle calcule $x_i = P(t_i)$, qu'elle transmet à B_i.

1. Écrire une fonction sage qui prend en entrées p, $(t_1, \ldots, t_n)$, s et rend en sortie $(x_1, \ldots, x_n)$.
2. Quel algorithme vu en cours permettra au groupe de calculer s à partir de leurs données $(x_1, \ldots, x_n)$, et des données publiques p et $(t_1, \ldots, t_n)$? Justifiez votre réponse.
3. Écrire une fonction sage prenant en entrées p, $(t_1, \ldots, t_n)$, $(x_1, \ldots, x_n)$, et retournant en sortie s.

4. Application numérique : calculez s sachant que : $p = 10007$, $n = 10$, $t_i = i$ pour $1 \leq i \leq n$ et $(x_1, \ldots, x_{10}) = [1707, 8016, 4310, 9802, 9049, 5879, 557, 5818, 3247, 7072]$.
5. Expliquez pourquoi un sous-ensemble strict des personnes B_i ne pourra pas obtenir d'information sur s à partir de leurs données et des données publiques (on pourra montrer par exemple que tout autre secret s' peut conduire pour au moins un aléa aux mêmes parts x_i détenues par les membres du sous-ensemble strict des B_i).

Exercice 3 Soit $F = \mathbb{Z}/17\mathbb{Z}$ et soit $Q \in F[x]$ le polynôme de degré 18 dont les coefficients rangés par degré croissant sont :

$$[13, 3, 9, 10, 1, 8, 4, 16, 13, 4, 8, 16, 16, 1, 11, 12, 5, 3, 1]$$

Le but de l'exercice est de factoriser Q grâce à l'algorithme de Cantor-Zassenhaus.

1. Q est un polynôme sans facteur carré, produit de trois polynômes irréductibles sur F de degrés 6. Cette affirmation peut être vérifiée par le calcul de quatre pgcd de polynômes ; lesquels ? justifiez votre réponse et faites ces calculs dans Sage.
2. Expliquez pourquoi le quotient $F[x]/(Q)$ est le produit direct de trois copies du corps fini F_{17^6}.
3. Rappelez pourquoi, si $a \in F_{17^6}^*$, alors $a^{\frac{17^6-1}{2}} \in \{1, -1\}$.
4. On propose l'algorithme suivant :

   Algorithme 1 [FACTORISATION DE Q]
   Entrée : Q.
   Sortie : Un facteur irréductible de Q de degré 6 ou "échec"
   1. Choisir au hasard $A \in F[x]$, $1 \leq \deg(A) \leq 17$
   2. Calculer $D = \text{pgcd}(A, Q)$. Si $\deg(D) = 6$, sortir D. Si $\deg(D) = 12$, sortir Q/D.
   3. Calculer $R = A^{\frac{17^6-1}{2}} \mod Q$.
   4. Si $R = 1$ ou $R = -1$, sortir "échec".
   5. Calculer $D = \text{pgcd}(R - 1, Q)$. Si $\deg(D) = 6$, sortir D. Si $\deg(D) = 12$, sortir Q/D.

   Expliquez pourquoi cet algorithme sort avec une probabilité supérieure à $0,75$ un facteur irréductible de Q de degré 6. Effectuez-le dans Sage plusieurs fois pour obtenir les trois facteurs irréductibles de Q.

Exercice 4 Soit $g = x^2 + 2y^2 - 3$ et $h = x^2 + xy + y^2 - 3$ deux polynômes de $\mathbb{Q}[x, y]$. Soit I l'idéal de $\mathbb{Q}[x, y]$ engendré par g et h. On munit $\mathbb{Q}[x, y]$ de l'ordre lexicographique tel que $x > y$.

1. Montrez à la main que (g, h) n'est pas une base de Groebner de I.
2. Soit B la base de Groebner réduite de I, calculez B avec Sage.
3. A l'aide de cette base, calculez l'ensemble $V(I)$ des solutions dans $\mathbb{C}$ du système suivnt (vous expliquerez votre démarche).

$$\begin{cases} x^2 + 2y^2 = 3 \\ x^2 + xy + y^2 = 3 \end{cases}$$

Vous devez trouver 4 points.

4. Les monômes standards sont les monômes de $\mathbb{Q}[x,y]$ qui ne sont divisibles par aucun des termes dominants des éléments de B. Déterminez les monômes standards de B et expliquez pourquoi ils forment une base du $\mathbb{Q}$-espace vectoriel $\mathbb{Q}[x,y]/I$. Quelle est la dimension de ce quotient et comment se compare-t-elle au cardinal de $V(I)$?

Exercice 5 Dans le cas de l'exercice 4, on constate que le cardinal de $V(I)$ est égal à la dimension du quotient $\mathbb{C}[x,y]/I$ (qui est ici la même que celle de $\mathbb{Q}[x,y]/I$). Vous allez maintenant démontrer cette propriété dans un cadre général.

Soit K un corps et soit I un idéal de $K[x,y]$. On note $V(I)$ l'ensemble des zéros de I :

$$V(I) = \{p = (a,b) \in K^2 \mid f(a,b) = 0 \text{ pour tout } f \in I\}$$

On suppose que $V(I)$ est fini et on note $V(I) = \{p_1, \ldots, p_n\}$ ses éléments. On suppose que I vérifie la propriété suivante :

$$(R) \text{ Pour tout } f \in K[x,y], \quad \text{si } \forall p_i \in V(I),\ f(p_i) = 0, \text{ alors } f \in I.$$

Un idéal qui vérifie (R) est dit *radical.*

1. Construire un polynôme $P_1 \in K[x,y]$ tel que $P_1(p_1) = 1$ et $P_1(p_i) = 0$ pour tout $i \geq 2$. (indication : puisque $p_i \neq p_1$, ils diffèrent en l'une des deux coordonnées..).
2. Déduire de 1. qu'il existe n polynômes $P_1, \ldots, P_n$ tels que pour tout $i \neq j$, $1 \leq i,j \leq n$, $P_i(p_i) = 1$ et $P_i(p_j) = 0$.
3. Montrez que ces n polynômes sont K-linéairement indépendants modulo I.
4. Soit $g \in K[x,y]$. Montrez que $g - \sum_{k=1}^{n} g(p_k)P_k$ appartient à I (utilisez (R)).
5. Déduire des questions 3. et 4. que $n = \dim(K[x,y]/I)$.

université de BORDEAUX	**Année Universitaire 2022–2023, Master CSI** DS, Théorie de la complexité (UE 4TCY701U) 16 novembre 2022, 9h30–12h30	Collège Sciences et Technologies

La rédaction doit être précise et concise. Le barème est indicatif.
Seules, les réponses correctement justifiées apporteront des points.

Exercice 1 : Applications du cours (3 points)

Répondez aux questions suivantes en quelques lignes, en justifiant clairement vos réponses.

1. Pour chacun des deux problèmes suivants, dire s'il est décidable et s'il est semi-décidable.

   (a) **Problème 1 : accessibilité d'un état.**
   Entrée : Une machine de Turing M (donnée par son code) et un état q de M.
   Question : La machine M a-t-elle un calcul sur le mot vide qui atteint l'état q ?

   (b) **Problème 2 : inclusion dans a^*.**
   Entrée : Une machine de Turing M sur l'alphabet d'entrée $\{a, b\}$.
   Question : La machine M accepte-t-elle uniquement des mots n'ayant que des a ?

2. Pour une machine de Turing M, on note $c(M)$ son code et on considère le langage suivant :

$$L = \{\, c(M) \mid M \text{ est une machine non déterministe qui } \textbf{n'accepte pas } c(M) \text{ en } 2^{|c(M)|} \text{ pas de calcul ou moins} \,\}.$$

   Le langage L est-il dans **NEXPTIME** ? Dans **NP** ? Que pouvez-vous en conclure ?

J'ai l'impression que $L \in$ NEXPTIME et $L \notin$ NP ce qui prouve NEXPTIME $\neq$ NP

Exercice 2 : P vs. NP-complet (6 points)

Pour chacun de ces problèmes, démontrer soit qu'il est **NP**-complet, soit qu'il est dans **P**.

Remarques

- Les réductions sont *simples* en choisissant le bon problème de départ.
- Pour les réductions, vous pouvez utiliser *tout* problème **NP**-complet vu en cours ou TD.
- Pour montrer qu'un problème est **NP**-complet, pensez à vérifier qu'il est dans **NP**.

On rappelle qu'en logique propositionnelle, un littéral est une variable ou une négation de variable, une clause est une disjonction de littéraux, et une formule CNF est une conjonction de clauses. Dans les problèmes suivants, les entiers donnés en entrée sont codés en binaire.

1. $\textsc{Sat}_{\geqslant 42}$

   Entrée : Une formule propositionnelle φ en forme CNF.
   Question : Existe-t-il une affectation des variables pour laquelle chaque clause de φ a au moins 42 littéraux à « vrai » ?

2. **FULL SAT**

Entrée : Une formule propositionnelle φ en forme CNF.
Question : Existe-t-il une affectation des variables pour laquelle chaque clause de φ a tous ses littéraux à « vrai » ?

3. **SAT MODIFIÉ**

Entrée : Une formule propositionnelle φ en forme **DNF** (c'est-à-dire, une disjonction de conjonctions de littéraux).
Question : Existe-t-il une affectation des variables qui rend la formule φ vraie et une autre affectation des variables qui rend la formule φ fausse ?

Dans un graphe non orienté dont les sommets sont coloriés, on dit qu'une arête est **mal coloriée** lorsque les deux sommets à ses extrémités ont la même couleur.

4. **3-COLORATION À UNE ERREUR PRÈS**

Entrée : Un graphe fini non-orienté G.
Question : Existe-t-il une coloration des sommets avec trois couleurs telle qu'il existe au plus une arête mal coloriée ?

5. **COUVERTURE PAR SOUS-ENSEMBLES**

Entrée : Un ensemble fini E, des sous-ensembles $E_1, \ldots, E_n$ de E, et un entier k.
Question : Existe-t-il k sous-ensembles parmi $E_1, \ldots, E_n$ dont l'union est égale à E ?

Exercice 3 : Machines à piles et à file (12 points)

On s'intéresse à des machines utilisant une mémoire et acceptant des mots. On considère d'abord les *machines à deux piles*. Le mot d'entrée d'une telle machine est initialement écrit dans une partie de sa mémoire. Cette mémoire est constituée de deux mots, qu'elle peut modifier en fonction de ses transitions. Chacun de ces deux mots représente une pile : les lettres de chaque mot sont les symboles stockés dans la pile correspondante, le haut de pile étant la première lettre du mot.

Formellement, une machine à deux piles est un tuple $\mathscr{P} = (\Sigma, \Gamma, Q, I, F, \alpha_1, \beta_1, \alpha_2, \beta_2)$ où où Σ est l'alphabet d'entrée, Γ l'alphabet de la mémoire, Q un ensemble fini d'états, $I \subseteq Q$ l'ensemble d'états initiaux et $F \subseteq Q$ l'ensemble d'états finaux. De plus, $\alpha_1, \beta_1, \alpha_2, \beta_2$ sont des sous-ensembles de $Q \times \Gamma \times Q$. Intuitivement, α_1 et β_1 sont les transitions agissant sur la première pile, et α_2 et β_2 sont celles agissant sur la seconde. De plus, α_1, α_2 sont les transitions d'empilement, et β_1, β_2 celles de dépilement.

On définit à présent le langage accepté par une machine à deux piles $\mathscr{P}$. Une *configuration* de $\mathscr{P}$ est un triplet $(q, x_1, x_2) \in Q \times \Gamma^* \times \Gamma^*$. Intuitivement, x_1 est le contenu courant de la première pile et x_2 est celui de la seconde. On définit une relation "$\to$" entre les configurations :

Dépilement 1 : Si $(q, a, r) \in \alpha_1$, alors pour tous $x_1, x_2 \in \Gamma^*$, on a $(q, a x_1, x_2) \to (r, x_1, x_2)$.

Dépilement 2 : Si $(q, a, r) \in \alpha_2$, alors pour tous $x_1, x_2 \in \Gamma^*$, on a $(q, x_1, a x_2) \to (r, x_1, x_2)$.

Empilement 1 : Si $(q, a, r) \in \beta_1$, alors pour tous $x_1, x_2 \in \Gamma^*$, on a $(q, x_1, x_2) \to (r, a x_1, x_2)$.

Empilement 2 : Si $(q, a, r) \in \beta_1$, alors pour tous $x_1, x_2 \in \Gamma^*$, on a $(q, x_1, x_2) \to (r, x_1, a x_2)$.

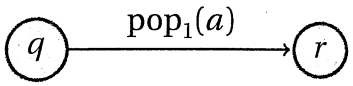
Enfin, on note $\xrightarrow{*}$ la relation définie par $(q,u,x)\xrightarrow{*}(r,v,y)$ si et seulement si il existe une suite de configurations deux à deux reliées par $\to$ allant de (q,u,x) à (r,v,y). Autrement dit, $(q,u,x)\xrightarrow{*}(r,v,y)$ lorsqu'il existe $n\in\mathbb{N}$ et $(q_0,u_0,x_0),\dots,(q_n,u_n,x_n)\in Q\times\Gamma^*\times\Gamma^*$ tels que,

$$(q,u,x)=(q_0,u_0,x_0)\to(q_1,u_1,x_1)\to\cdots\to(q_n,u_n,x_n)=(r,v,y).$$

Notez qu'on a toujours $(q,u,x)\xrightarrow{*}(q,u,x)$: c'est le cas $n=0$. Le langage accepté par $\mathscr{P}$, noté $L(\mathscr{P})$, est l'ensemble des mots $w\in\Sigma^*$ tels qu'il existe $q\in I$ et $r\in F$ qui satisfont $(q,w,\varepsilon)\xrightarrow{*}(r,\varepsilon,\varepsilon)$. En particulier, les deux piles doivent être vides à la fin de l'exécution.

I. Machines à deux piles sans restriction.

1. Construire une machine à deux piles (en donnant la liste de ses transitions) sur l'alphabet d'entrée $\Sigma=\{a,b\}$ acceptant le langage $\{a^nb^n \mid n\geqslant 0\}$, ainsi qu'une explication des phases de son exécution.

> **Remarque**
>
> Vous pouvez utiliser une notation graphique indiquant la nature de l'opération et la lettre concernée. Par exemple, vous pouvez indiquer que $(q,a,r)\in\alpha_1$ par le dessin :
>
> 
>
>
> et utiliser des notations similaires pour α_2,β_1 et β_2 (comme pop_2, push_1 et push_2).

2. Expliquer en français, sans donner les transitions, comment construire une machine à deux piles M sur l'alphabet d'entrée $\Sigma=\{a,b,c\}$ telle que $L(M)=\{a^nb^nc^n \mid n\geqslant 0\}$.
3. Montrer que toute machine à deux piles peut être simulée par une machine de Turing. Autrement dit, on demande d'expliquer comment, à partir du code source d'une machine à deux piles, on peut construire une machine de Turing acceptant le même langage.

> **Indication**
>
> Pensez à utiliser une machine de Turing à plusieurs bandes.

4. Montrer que toute machine de Turing à une bande peut être simulée par une machine à deux piles. Autrement dit, on demande d'expliquer comment, à partir du code source d'une machine de Turing, on construit une machine à deux piles acceptant le même langage.

II. Machines restreintes à deux piles.

On dit qu'une machine à deux piles $\mathscr{P}=(\Sigma,\Gamma,Q,I,F,\alpha_1,\beta_1,\alpha_2,\beta_2)$ est *restreinte* lorsque β_1 est vide : la machine n'empile jamais sur la première pile. Cette première pile ne sert donc qu'à lire le mot d'entrée (par des opérations « Dépilement 1 »).

5. Étant donnée une *grammaire hors-contexte* $\mathscr{G}$, décrire une *machine restreinte à deux piles* $\mathscr{P}$ telle que $L(\mathscr{G})=L(\mathscr{P})$. Vous pouvez utiliser les symboles de $\mathscr{G}$ dans la pile 2.
6. Inversement, étant donnée une *machine restreinte à deux piles* $\mathscr{P}$, décrire une *grammaire hors-contexte* $\mathscr{G}$ telle que $L(\mathscr{G})=L(\mathscr{P})$.

Indication

Pour toute paire d'états (q, r) de $\mathscr{P}$, la grammaire $\mathscr{G}$ contient une variable $X_{q,r}$, telle qu'un mot $w \in \Sigma^*$ peut être dérivé depuis celle-ci si et seulement si $(q, w, \varepsilon) \xrightarrow{*} (r, \varepsilon, \varepsilon)$. Vous pouvez distinguer deux cas, selon que le calcul revient de façon intermédiaire à une pile 2 vide, ou non.

III. Machines à file.

On définit maintenant une autre sorte de machine : les machines à file. Formellement, une machine à file est donnée par un tuple $\mathscr{F} = (\Sigma, \Gamma, Q, I, F, \alpha_1, \alpha_2, \gamma_2)$ où $\Sigma, \Gamma, Q, I, F, \alpha_1, \alpha_2$ sont définis comme pour les machines à deux piles, et $\gamma_2 \subseteq Q \times \Gamma \times Q$ représente les *transitions d'enfilement*. La machine peut donc lire (par dépilement) sur la mémoire 1 ou 2 (via α_1 ou α_2). Grâce à γ_2, elle peut enfiler sur le second mot de la mémoire. Formellement, une configuration de $\mathscr{F}$ est à nouveau un triplet $(q, x_1, x_2) \in Q \times \Gamma^* \times \Gamma^*$, et la relation "$\rightarrow$" entre configurations est maintenant définie ainsi (les dépilements sont exactement les mêmes que précédemment) :

Dépilement 1 : Si $(q, a, r) \in \alpha_1$, alors pour tous $x_1, x_2 \in \Gamma^*$, on a $(q, a x_1, x_2) \rightarrow (r, x_1, x_2)$.

Dépilement 2 : Si $(q, a, r) \in \alpha_2$, alors pour tous $x_1, x_2 \in \Gamma^*$, on a $(q, x_1, a x_2) \rightarrow (r, x_1, x_2)$.

Enfilement 2 : Si $(q, a, r) \in \gamma_2$, alors pour tous $x_1, x_2 \in \Gamma^*$, on a $(q, x_1, x_2) \rightarrow (r, x_1, x_2 a)$.

Notez que, comme les machines *restreintes* à pile, une machine à file ne peut pas ajouter de lettre sur le premier mot de la mémoire, x_1. Notez aussi que contrairement aux opérations d'empilement des machines à deux piles, la lettre insérée par enfilement sur le second mot de la mémoire, x_2, est ajoutée en **fin** de mot.

On définit $\xrightarrow{*}$ comme précédemment, et le langage accepté par $\mathscr{F}$, noté $L(\mathscr{F})$, est l'ensemble des mots $w \in \Sigma^*$ tels qu'il existe $q \in I$ et $r \in F$ qui satisfont $(q, w, \varepsilon) \xrightarrow{*} (r, \varepsilon, \varepsilon)$.

7. Expliquer à haut niveau le principe d'une machine à file acceptant l'ensemble des mots sur l'alphabet d'entrée $\Sigma = \{a, b, c\}$ dont les nombres de a, de b et de c sont égaux (sans donner explicitement la liste des transitions).
8. Montrer que toute machine à deux piles (même non restreinte) peut être simulée par une machine à file. Autrement dit, expliquer comment, à partir du code source d'une machine à deux piles, on peut construire une machine à file acceptant le même langage.

IV. Questions récapitulatives.

Pour chacune des affirmations suivantes, dites si elle est vraie ou fausse en justifiant brièvement la réponse. Vous pouvez utiliser les résultats des questions précédentes, même si vous ne les avez pas faites.

9. Tout langage accepté par une machine restreinte à deux piles est régulier.
10. Étant donnée une machine à deux piles $\mathscr{P}$, le problème de savoir si un mot d'entrée appartient à $L(\mathscr{P})$ est dans **NP**.
11. Étant données deux machines restreintes à deux piles $\mathscr{P}_1$ et $\mathscr{P}_2$, le problème de savoir si $L(\mathscr{P}_1) \cap L(\mathscr{P}_2) = \emptyset$ n'est pas semi-décidable.
12. Étant donnée une machine à file $\mathscr{F}$, le problème de savoir si $\mathscr{F}$ accepte le mot vide est dans **P**.

université de BORDEAUX	**Année Universitaire 2022–2023, Master CSI** DS, Théorie de la complexité (UE 4TCY701U) 12 décembre 2022, 14h30–17h30	Collège Sciences et Technologies

Le barème est indicatif. Vous pouvez utiliser le résultat de toute question, même non traitée. La rédaction doit être précise. Seules, les réponses correctement justifiées apportent des points.

Exercice 1 : Applications du cours (3 points)

Répondez aux questions suivantes en quelques lignes, en justifiant clairement vos réponses.

1. On considère le langage des codes des machines de Turing qui acceptent exactement les multiples de 42 (codés en base 2). Le langage L est-il décidable? semi-décidable? dans **P**?
2. Tout problème de la classe **PSPACE** se réduit-il au problème de correspondance de Post?
3. Si L est un **NP**-difficile, est-il vrai que tout langage K contenant L est aussi **NP**-difficile?

Exercice 2 : Problèmes NP-complets, NL-complets, dans P ou dans L (7 points)

Montrer que chacun des problèmes suivants est soit **NP**-complet, soit **NL**-complet, soit dans **P**, soit dans **L**, en donnant à chaque fois la plus petite classe possible contenant le problème. Dans tous les problèmes, les entiers donnés en entrée sont codés en binaire.

1. Arbre de poids fixé
   Entrée : Un entier $P \in \mathbb{N}$, un graphe $G = (V, E)$ non orienté, et une fonction $p : E \to \mathbb{N}$.
   Question : Existe-t-il un sous-ensemble $F \subseteq E$ des arêtes de G tel que (V, F) est un arbre (c'est-à-dire un graphe connexe sans cycle) et $\sum_{e \in F} p(e) = P$?
2. Chemin Quasi-Hamiltonien
   Entrée : Un graphe orienté G.
   Question : Le graphe G possède-t-il un chemin qui visite chaque sommet au moins une fois et au plus deux fois?
3. Forte connexité
   Entrée : Un graphe **orienté** G.
   Question : Le graphe G est-il fortement connexe, c'est-à-dire, est-il vrai que pour tous sommets s et t, il y a un chemin de s à t ?
4. Not-All-Equal 4Sat
   Entrée : Une formule 4-CNF (c'est-à-dire une conjonction de clauses, chacune du type $(\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4)$, où chaque ℓ_i est un littéral).
   Question : Y a-t-il une affectation des variables pour laquelle chaque clause a au moins un littéral vrai et au moins un littéral faux?
5. Facteur Commun
   Entrée : Un alphabet Σ, deux mots $u, v \in \Sigma^*$ et un entier $n \in \mathbb{N}$.
   Question : Existe-t-il un mot $w \in \Sigma^*$ de longueur au moins n qui est facteur de u et v (c'est-à-dire que $u = u_1 w u_2$ et $v = v_1 w v_2$) ?

6. PARTITION EN 3 CLIQUES

Entrée : Un graphe non orienté $G = (V, E)$.

Question : Existe-t-il une partition de l'ensemble V des sommets de G en trois parties deux à deux disjointes V_1, V_2 et V_3, avec $V = V_1 \cup V_2 \cup V_3$, telle que pour chaque $i = 1, 2, 3$, le sous-graphe de G construit en gardant les sommets de V_i et les arêtes de G reliant les points de V_i soit une clique (c'est-à-dire que les sommets de V_i sont deux à deux reliés par une arête dans G) ?

Un ***semigroupe*** est un ensemble S muni d'une multiplication associative $(x, y) \mapsto xy$: pour tous $x, y, z \in S$, on a $(xy)z = x(yz)$. On peut ainsi définir la puissance n-ème d'un élément $s \in S$ sans se préoccuper du parenthésage : s^n désigne la multiplication de n copies de s, c'est-à-dire $\underbrace{ss\cdots s}_{n \text{ fois}}$. On dit qu'un semigroupe S est *42-périodique* si pour tout élément $s \in S$, il existe un entier $n \geqslant 1$ tel que $s^n = s^{n+42}$.

7. SEMIGROUPE 42-PÉRIODIQUE

Entrée : Un semigroupe S donné par sa table de multiplication.

Question : Le semigroupe S est-il 42-périodique ?

Exercice 3 : Problèmes P-complets (8 points)

En logique propositionnelle, on appelle *clause de Horn* une clause qui contient ***au plus un*** littéral positif. Par exemple, les trois clauses suivantes sont des clauses de Horn : (v), $(\neg v \vee \neg x)$ et $(\neg y \vee \neg z \vee \neg x \vee \neg u \vee v)$. À l'inverse, la clause suivante n'est **pas** une clause de Horn car elle contient deux littéraux positifs (soulignés) : $(\neg y \vee \underline{z} \vee \neg x \vee \neg u \vee \underline{v})$.

1. Montrer que le problème HORN-SAT ~~suivant est dans P en décrivant en français un algorithme~~ polynomial.

   Entrée : Une conjonction $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$ de clauses de Horn.

   Question : La formule φ est-elle satisfaisable ?

   Vous pourrez d'abord considérer le cas où toute clause a au moins un littéral négatif. Sinon, votre algorithme pourra initialiser toutes les variables à « Faux », puis, modifier itérativement cette affectation selon la formule φ, jusqu'à obtenir une conclusion. Justifiez précisément la complexité de l'algorithme obtenu.

2. On se place sur l'alphabet $\{0, 1\}$. Soit $A \subseteq \{0, 1\}^*$ un problème de la classe **P**. On suppose que $A \neq \emptyset$ et $A \neq \{0, 1\}^*$. Quels sont les problèmes de **P** se réduisant polynomialement à A ?

On définit maintenant la notion de **P**-complétude. On dit qu'un problème A est **P**-*complet* si les deux conditions suivantes sont vérifiées :

(a) $A \in \mathbf{P}$,

(b) tout problème de **P** se réduit à A par une réduction en espace logarithmique.

3. Montrer que le problème HORN-SAT est **P**-complet.

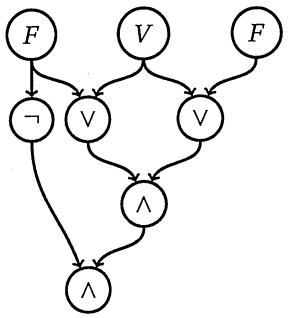
   Remarque. Nous n'avons encore rencontré aucun problème **P**-complet. Cela donne une indication sur ce qu'il faut faire dans cette question.

4. Montrer que le problème reste **NP**-complet si les clauses de la formule d'entrée contiennent au plus 3 littéraux.

On passe maintenant à un second problème. Un ***circuit Booléen*** est un **graphe *orienté sans circuit*** dont les sommets sont appelés des *portes*. Les portes sont étiquetées soit par des valeurs Booléennes, Vrai (V) ou Faux (F), soit par des connecteurs logiques ($\wedge$ pour **Et**, $\vee$ pour **Ou** et $\neg$ pour **Non**). Un circuit doit satisfaire les propriétés suivantes :

- Les portes qui n'ont pas d'arc entrant (appelées *portes d'entrée*) sont étiquetées par une valeur Booléennes : Vrai (V) ou Faux (F).
- Les autres portes sont étiquetées par des connecteurs logiques ($\wedge$, $\vee$ ou $\neg$). Une porte étiquetée par $\neg$ a un unique arc entrant, et une porte étiquetée par $\wedge$ ou $\vee$ a exactement 2 arcs entrants.
- Il y a une unique porte sans arc sortant, appelée la *porte résultat*. Les autres portes peuvent avoir un nombre arbitraire d'arcs sortants.

Un exemple de circuit Booléen est donné ci-dessous. La porte résultat, en bas, est étiquetée $\wedge$.



Chaque porte calcule une valeur Booléenne. Sauf pour la porte résultat, cette valeur est ensuite transmise le long de tous ses arcs qui sortent de la porte. Comme les portes peuvent avoir plusieurs arcs sortants (sauf la porte résultat), une même sortie peut ainsi être utilisée plusieurs fois en entrée. Intuitivement, une porte effectue sur ses arcs entrants l'opération logique correspondant à son étiquette :

- Une porte d'entrée calcule Vrai si son étiquette est V et Faux si son étiquette est F.
- Une porte étiquetée par $\neg$, et dont l'unique arc entrant transmet la valeur val, calcule $\neg val$.
- Une porte étiquetée par $\vee$, et dont les deux arc entrants transmettent des valeurs val_1 et val_2, calcule $val_1 \vee val_2$.
- Une porte étiquetée par $\wedge$, et dont les deux arcs entrants transmettent des valeurs val_1 et val_2, calcule $val_1 \wedge val_2$.

La *valeur calculée par le circuit* est celle de sa porte résultat. On considère le problème ÉVALUATION DE CIRCUIT suivant :

Entrée : Un circuit Booléen C.
Question : Est-ce que le circuit calcule la valeur Booléenne *Vrai*?

5. Montrer que ÉVALUATION DE CIRCUIT est **P**-complet. Pour montrer la **P**-difficulté, vous pouvez utiliser la question 4 même si vous ne l'avez pas faite.
6. On considère maintenant une restriction, appelée ÉVALUATION DE CIRCUIT MONOTONE :

   Entrée : Un circuit Booléen C **sans porte étiquetée** $\neg$
   Question : Le circuit C calcule-t-il la valeur *Vrai*?

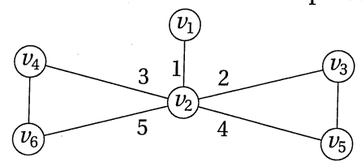
   Montrer que ÉVALUATION DE CIRCUIT MONOTONE est encore **P**-complet.

Exercice 4 : Cycles dans les graphes non orientés (5 points)

Dans cet exercice, on considère des graphes ***non orientés***. Étant donné un tel graphe G, un *cycle dans* G est une suite finie d'arêtes $(e_1, e_2, \ldots, e_k)$ consécutives, telle que pour tout i, $e_i \neq e_{i+1}$, et revenant au sommet de départ. Ainsi, e_i et e_{i+1} partagent exactement un sommet. Notez que si e est une arête, (e, e) ne constitue pas un cycle (à cause de la condition $e_i \neq e_{i+1}$). On considère le problème CYCLE NON ORIENTÉ :

Entrée : Un graphe *non-orienté* G.
Question : Existe-t-il un cycle dans G ?

On veut montrer que CYCLE NON ORIENTÉ est dans **L**. On suppose dans un premier temps qu'avec le graphe d'entrée $G = (V, E)$, on dispose aussi pour chaque sommet $v \in V$ d'une numérotation 1, 2,..., d_v des arêtes incidentes au sommet v (où d_v est le nombre de telles arêtes). Par exemple, sur le graphe non orienté suivant, les 5 arêtes incidentes à v_2 sont numérotées de 1 à 5. On a une numérotation des arêtes incidentes à chaque autre sommet du graphe, mais sur la figure, cette numérotation n'est représentée que pour v_2.



On considère un graphe non orienté $G = (V, E)$. Pour chaque sommet $v \in V$ et chaque arête e incidente à v, on définit le parcours (infini) $P(v, e)$ de G suivant. On part du sommet v et on prend l'arête e. Ensuite, à chaque fois qu'on arrive dans un nouveau sommet $u \in V$, si on est entré par l'arête incidente numérotée par k, on prend ensuite celle numérotée par $k+1$ (si k était l'arête incidente maximale de u, on prend celle numérotée par 1).

On dit que $P(v, e)$ est un *bon parcours* si il repasse par v et que la première fois que cela arrive, l'arête incidente utilisée pour entrer dans v est différente de e.

1. On considère un sommet $v \in V$ et e une arête incidente à v. Montrer que si $P(v, e)$ est un bon parcours, alors G contient un cycle.
2. Montrer que si G contient un cycle, alors il existe un sommet $v \in V$ et une arête $e \in E$ incidente à v tels que $P(v, e)$ est un bon parcours.
3. Déduire des questions 1 et 2 que le problème CYCLE NON ORIENTÉ, restreint aux entrées pour lesquelles on a une numérotation des arêtes incidentes à chaque sommet, est dans **L**.
4. Expliquer pourquoi l'hypothèse qu'on dispose d'une telle numérotation n'est ***pas nécessaire*** pour obtenir un algorithme dans **L**, autrement dit, pourquoi le problème original CYCLE NON ORIENTÉ est lui-même dans **L**.
5. On considère maintenant la version de ce problème dans un graphe orienté. Dans un tel graphe, un cycle orienté est une suite finie d'arêtes $e_1, \ldots, e_k$ deux à deux distinctes, consécutives, telles que $e_i = s_i \to s_{i+1}$, avec $s_{k+1} = s_1$ (le cycle revient à son point de départ, comme en version non orientée). Donnez la complexité du problème suivant, en justifiant :

   Entrée : Un graphe ***orienté*** G.
   Question : Existe-t-il un cycle dans G ?