MP2I
2021/2022

Informatique
Devoir surveillé 2
13 novembre 2021

C I V

Durée totale 4h.
Tout document interdit. Tout dispositif électronique interdit.
On vous demande d'écrire lisiblement et de bien présenter. Les programmes en particulier doivent être bien indentés et lisibles. Utilisez de préférence plusieurs couleurs. **Commentez** votre code dès que ce qu'il fait n'est pas évident.

1 Manipulation de sous-ensembles de $[\![0, n-1]\!]$

On va ici supposer définie une constante entière $n > 0$ accessible directement en `C` (on peut donc écrire `int t[n];` pour déclarer un tableau de cette taille).
L'objectif de ce problème est de réaliser une bibliothèque de manipulation de sous-ensembles de $E = [\![0, n-1]\!] = \{0, 1, 2, \ldots, n-1\}$.
Pour cela, on va les représenter sous la forme de tableaux de booléens (on suppose `stdbool.h` inclus). Si $S \subset E$ on le représente par le tableaux `t` de n booléens tel que $\forall i \in E, i \in S \iff$ `t[i] = true`.
Ainsi, le sous-ensemble $\{1, 3, 5\}$ de $[\![0, 5]\!]$ est représenté par le tableau

`i`	0	1	2	3	4	5
`t[i]`	false	true	false	true	false	true

Dans la suite du sujet on utilise le mot **sous-ensemble** pour désigner un tel tableau représentant un sous-ensemble de `E`.
Convention d'appel : comme les sous-ensembles seront tous représentés par des tableaux de taille n, il n'est pas nécessaire de passer leur taille en argument. On écrira donc juste `f(bool *t)` pour une fonction prenant un tableau, `f(bool *t1, bool *t2)` pour une en prenant deux, et ainsi de suite.
Remarque : on vous laisse ici souvent libre des arguments, des valeurs de retour, de la nomenclature de votre fonction. En contrepartie, on attend que la déclaration des fonctions soit brièvement documentée.

Q01 Écrire une fonction qui, étant donné un sous-ensemble `S` de `E` et un élément `x`, renvoie `true` si et seulement si $x \in S$.

Q02 Écrire une fonction qui renvoie le nombre d'élément d'un sous-ensemble.

Q03 Écrire une fonction `void complementaire(bool *e, bool *ce);` qui prend en entrée un sous-ensemble `e` et place son complémentaire dans `ce`.

Q04 Prouver la correction de la fonction précédente.

Q05 Écrire une fonction `bool disjoint(bool *e1, bool *e2);` qui prend en entrée deux sous-ensembles `e1` et `e2`, puis renvoie `true` si et seulement si les sous-ensembles sont disjoints : ils n'ont aucun éléments en commun.

Q06 Écrire une fonction `void union(bool *e1, bool *e2, bool *ue);` qui prend en entrée deux sous-ensembles `e1` et `e2`, puis place leur union dans `ue`.

Q07 Écrire une fonction `void intersection(bool *e1, bool *e2, bool *ie);` qui prend en entrée deux sous-ensembles `e1` et `e2`, puis place leur intersection dans `ie`.

Q08 Écrire une fonction `void difference(bool *e1, bool *e2, bool *de);` qui prend en entrée deux sous-ensembles `e1` et `e2`, puis place la différence `e1\e2` dans `de`.

On note $I = [\![0, 2^{n-1} - 1]\!]$. Si $S \subset [\![0, n-1]\!]$, on pose $u(S) = \sum_{i \in S} 2^i$.

Q09 Écrire une fonction `int ensemble_vers_entier(bool *e);` qui prend en entrée un sous-ensemble e et renvoie l'entier $u(e)$.

Q10 Écrire une fonction `void entier_vers_ensemble(int u, bool *e);` qui prend en entrée un entier u et place dans e le sous-ensemble tel que $u = u(e)$.

On dit qu'une suite finie $F_0, \dots, F_{p-1}$ de sous-ensembles de E est une couverture de E si et seulement si

$$\forall i \in E, \exists j \in [\![0, p-1]\!], i \in F_j$$

Si, de plus, les F_j sont deux à deux disjoints, on dit qu'ils forment une partition de E.

Q11 Écrire une fonction `bool couverture(int *f, int p);` qui prend en entrée un tableau de p entiers `f` où pour tout $i \in [\![0, p-1]\!]$, `f[i]` $= u(F_i)$ avec F_i un sous-ensemble de E, et renvoie un booléen indiquant si $(F_0, \dots, F_{p-1})$ est une couverture de E.
Indications : on pourra soit garder les ensembles sous forme d'entiers, soit passer par des représentations temporaires en tableau en tirant partie du fait que n est constant pour les déclarer en temps que variables locales.

Q12 Prouver la correction de la fonction précédente.

Q13 Écrire une fonction `bool partition(int *f, int p);` qui prend en entrée un tableau de p entiers `f` où pour tout $i \in [\![0, p-1]\!]$, `f[i]` $= u(F_i)$ avec F_i un sous-ensemble de E, et renvoie un booléen indiquant si $(F_0, \dots, F_{p-1})$ est une partition de E.

Q14 Prouver la correction de la fonction précédente.

2 Points fixes de fonctions à domaine fini

Dans ce problème, on s'intéresse aux points fixes des fonctions $f : E \to E$, où E est un ensemble fini. Le calcul effectif et efficace des points fixes de telles fonctions est un problème récurrent en informatique (transformation d'automates, vérification automatique de programmes, algorithmique des graphes, etc.), et admet différentes approches selon la structure de E et les propriétés de f.
On suppose par la suite un entier strictement positif $n > 0$ fixé et accessible comme une constante entière en `C` (on peut donc écrire `int t[n];` pour déclarer un tableau de cette taille) de même nom (dans la suite du sujet, on confondra l'entier n et la variable informatique `n`), et on pose $E_n = \{0, \dots, n-1\}$. On représente une fonction $f : E_n \to E_n$ par un tableau `t` de taille n, autrement dit $f(x) =$ `t[x]` pour tout $x = 0, \dots, n-1$. Ainsi la fonction f_0 qui à $x \in E_{10}$ associe $2x + 1$ modulo 10 est-elle représentée par le tableau `t` suivant :

`i`	0	1	2	3	4	5	6	7	8	9
`t[i]`	1	3	5	7	9	1	3	5	7	9

Le temps de calcul d'une fonction `f` de paramètres $p_1, \dots, p_k$ est défini comme le nombre d'opérations (accès en lecture ou écritures à une case d'un tableau ou à une variable, appel à une des primitives données dans l'énoncé) exécutées par `f` pour ces paramètres ; on note $T(f, n)$ le temps de calcul maximal pris sur tous les paramètres possibles pour n fixé. On dit que `f` s'exécute en temps linéaire s'il existe des réesl $\alpha, \beta > 0$ et un entier $N \geqslant 0$ tels que $\alpha n \leqslant T(f, n) \leqslant \beta n$ pour tout $n \geqslant N$. De même, on dit que `f` s'exécute en temps logarithmique s'il existe des réels $\alpha, \beta > 0$ et un entier $N \geqslant 0$ tels que $\alpha \log n \geqslant T(f, n) \geqslant \beta \log n$ pour tout $n \geqslant N$.

2.1 Recherche de point fixe : cas général

On rappelle que x est un point fixe de la fonction f si et seulement si $f(x) = x$.

Q15 Écrire une fonction `bool admet_point_fixe(int *t)` qui prend en argument un tableau `t` de taille n et renvoie `true` si la fonction $f : E_n \to E_n$ représentée par `t` admet un point fixe, `false` sinon. Par exemple, `admet_point_fixe` devra renvoyer `true` pour le tableau donné en introduction, puisque 9 est un point fixe de la fonction f_0 qui à x associe $2x + 1$ modulo 10.

Q16 Écrire une fonction `int nb_points_fixes(int *t)` qui prend en argument un tableau `t` de taille n et renvoie le nombre de points fixes de la fonction $f : E_n \to E_n$ représentée par `t`. Par exemple, `nb_point_fixes` devra renvoyer 1 pour le tableau donné en introduction, puisque 9 est le seul point fixe de f_0.

On note f^k l'itérée k-ième de f, autrement dit

$$\begin{aligned} f^k : E_n &\to E_n \\ x &\mapsto \underbrace{f(f(\dots f}_{k \text{ fois}}(x))\dots) \end{aligned}$$

Q17 Écrire une fonction `int itere(int *t, int x, int k)` qui prend en premier argument un tableau `t` de taille n représentant une fonction $f : E_n \to E_n$, en deuxième et troisième arguments des entiers x, k de E_n et renvoie $f^k(x)$.

Q18 Écrire une fonction `int nb_points_fixes_iteres(int *t, int k)` qui prend en premier argument un tableau `t` de taille n représentant une fonction $f : E_n \to E_n$, en deuxième argument un entier $k \geqslant 0$ et renvoie le nombre de points fixes de f^k.

Un élément $z \in E_n$ est dit *attracteur principal* de $f : E_n \to E_n$ si et seulement si z est un point fixe de f, et pour tout $x \in E_n$, il existe un entier $k \geqslant 0$ tel que $f^k(x) = z$.
Afin d'illustrer cette notion, on pourra vérifier que la fonction f_1 représentée par le tableau ci-dessous admet 2 comme attracteur principal.

`i`	0	1	2	3	4	5	6
`t[i]`	5	5	2	2	0	2	2

En revanche, on notera que la fonction f_0 donnée en introduction n'admet pas d'attracteur principal puisque $f_0^k(0) \neq 9$ quel que soit l'entier $k \geqslant 0$.

Q19 Écrire une fonction `bool admet_attracteur_principal(int *t)` qui prend en argument un tableau `t` de taille n et renvoie `true` si et seulement si la fonction $f : E_n \to E_n$ représentée par `t` admet un attracteur principal, `false` sinon. On ne requiert pas ici une solution efficace.

On suppose aux question 20 et 21 que f admet un attracteur principal. Le *temps de convergence* de f en $x \in E_n$ est le plus petit entier $k \geqslant 0$ tel que $f^k(x)$ soit un point fixe de f. Pour la fonction f_1 ci-dessus, le temps de convergence en 4 est 3. En effet, $f_1(4) = 0, f_1^2(4) = 5, f_1^3(4) = 2$, et 2 est un point fixe de f_1. On note $\texttt{tc}(f, x)$ le temps de convergence de f en x.

Q20 Écrire une fonction `int temps_de_convergence(int *t, int x)` qui prend en premier argument un tableau `t` de taille n représentant une fonction $f : E_n \to E_n$ qui admet un attracteur principal, en deuxième argument un entier x de E_n, et renvoie le temps de convergence de f en x. On pourra admettre qeu $\texttt{tc}(f, x)$ vaut 0 si x est un point fixe de f, et $1 + \texttt{tc}(f, f(x))$ si x n'est pas un point fixe de f.

Q21 Écrire une fonction `temps_de_convergence_max(t)` qui prend en argument un tableau `t` de taille n représentant une fonction $f : E_n \to E_n$ qui admet un attracteur principal et renvoie $\max_{i=0\dots n-1} \texttt{tc}(f, i)$. On impose un temps de calcul **linéaire** en la taille n du tableau [a]. À titre d'indication, on pourra au besoin créer des tableaux intermédiaire en tirant partie du fait que n est une constante.
On ne demande pas de démonstration du fait que le temps de calcul de la solution proposée est linéaire, mais il est impératif d'expliquer clairement le fonctionnement de votre fonction.

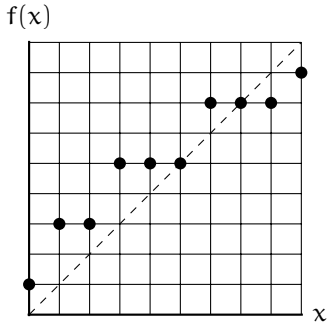
a. Une fonction ne respectant pas cette condition ne rapportera pas de point !

2.2 Recherche efficace de points fixes

Toute fonction `point_fixe(t)` retournant un point fixe d'une fonction arbitraire est de complexité au mieux linéaire en n. On s'intéresse maintenant à des améliorations possibles de cette complexité lorsque la fonction considérée possède certaines propriétés spécifiques. Nous examinons deux cas.

Premier cas

Le premier cas que nous considérons est celui d'une fonction croissante de E_n dans E_n. On rappelle qu'une fonction $f : E_n \to E_n$ est croissante si et seulement si pour tous $x, y \in E_n$ tels que $x \leqslant y, f(x) \leqslant f(y)$.
On admet qu'une fonction croissante de E_n dans E_n admet toujours un point fixe.
À titre d'exemple, la fonction dont le tableau et le graphe sont donnés ci-dessous est croissante. Elle a deux points fixes, à savoir les entiers 5 et 7.



i	0	1	2	3	4	5	6	7	8	9
t[i]	1	3	3	5	5	5	7	7	7	8

Q22 Écrire une fonction `bool est_croissante(int *t)` qui prend en argument un tableau `t` de taille n et renvoie `true` si la fonction représentée par `t` est croissante, et `false` sinon. On impose un temps de calcul **linéaire** en la taille n du tableau. On ne demande pas de démonstration du fait que le temps de calcul de la solution proposée est linéaire.

Q23 Écrire une fonction `int point_fixe(int *t)` qui prend en argument un tableau `t` de taille n représentant une fonction croissante $f : E_n \to E_n$, et retourne un entier $x \in E_n$ tel que $f(x) = x$. On impose un temps de calcul **logarithmique** en la taille n du tableau. On ne demande pas ici de démonstration du fait que le temps de calcul de la solution proposée est logarithmique, ceci étant le sujet de la question suivante.

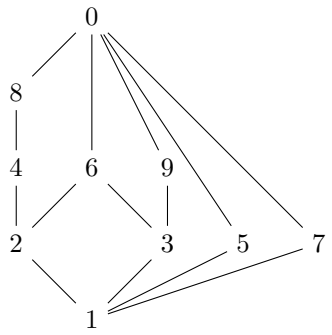
Q24 Démontrer que la fonction de la question 23 termine.

Deuxième cas

On peut généraliser la notion de fonction croissante comme suit. On rappelle qu'une relation binaire $\preccurlyeq$ sur un ensemble E est une relation d'ordre si et seulement si elle est réflexive ($x \preccurlyeq x$ pour tout $x \in E$), anti-symétrique (pour tous $x, y \in E$, si $x \preccurlyeq y$ et $y \preccurlyeq x$, alors $x = y$), et transitive (pour tous $x, y, z \in E$, si $x \preccurlyeq y$ et $y \preccurlyeq z$ alors $x \preccurlyeq z$). Soit $\preccurlyeq$ une relation d'ordre sur E. Une fonction $f : E \to E$ est *croissante au sens de* $\preccurlyeq$ si et seulement si pour tous $x, y \in E$, $x \preccurlyeq y$ implique $f(x) \preccurlyeq f(y)$.
Ceci généralise la notion de fonction croissante de E_n dans E_n, que l'on retrouve en prenant $E = E_n$ et $\preccurlyeq$ la relation d'ordre $\leqslant$. On s'intéresse dorénavant à d'autres relations d'ordre sur E_n.
On dit qu'un élément m de E est un *plus petit élément* de E au sens de $\preccurlyeq$ si et seulement si, pour tout $x \in E$, $m \preccurlyeq x$.
On admet que pour tout ensemble fini E, muni d'une relation d'ordre $\preccurlyeq$ et qui admet un plus petit élément m au sens de $\preccurlyeq$, pour toute fonction croisante $f : E \to E$ au sens de $\preccurlyeq$, il existe un entier $k \geqslant 0$ tel que $f^k(m)$ est un point fixe de f dans E.

Q25 Soit E un ensemble fini quelconque muni d'une relation d'ordre $\preccurlyeq$ et admettant un plus petit élément m au sens de $\preccurlyeq$. Soit $f : E \to E$ une fonction croissante au sens de $\preccurlyeq$, et soit $k \geqslant 0$ un entier tel que $f^k(m)$ soit un point fixe de f dans E. Démontrer que $f^k(m)$ est en fait le plus petit point fixe de f au sens de $\preccurlyeq$, autrement dit que pour tout autre point fixe x de f dans E, on a $f^k(m) \preccurlyeq x$.

Nous nous intéressons maintenant à un choix particulier d'ordre $\preccurlyeq$, appelé *ordre de divisibilité* et noté $|$. Précisément, on note $a|b$ la relation d'ordre « a divise b »sur les entiers positifs, vraie si et seulement s'il existe un entier $c \geqslant 0$ tel que $ca = b$. Ainsi l'ensemble E_{10} ordonné par divisibilité peut se représenter graphiquement comme suit :



D'après la définition donnée précédemment, une fonction $\mathtt{f} : \mathtt{E_n} \to \mathtt{E_n}$ *croissante au sens de l'ordre de divsibilité* est une fonction telle que pour tout $\mathtt{x}, \mathtt{y}$ dans $\mathtt{E_n}$, si $\mathtt{x}|\mathtt{y}$ alors $\mathtt{f}(\mathtt{x})|\mathtt{f}(\mathtt{y})$. Par exemple la fonction représentée par le tableau ci-dessous est croissante au sens de l'ordre de divisibilité.

`i`	0	1	2	3	4	5	6	7	8	9
`t[i]`	0	2	4	6	4	8	0	2	0	6

On remarque que, par la question 25, toute fonction de $\mathtt{E_n}$ dans $\mathtt{E_n}$ croissante au sens de l'ordre de divisibilité a un plus petit point fixe au sens de l'ordre de divisibilité.
On rappelle que le pgcd de deux entiers $\mathtt{x} \geqslant 1$ et $\mathtt{y} \geqslant 1$ est le plus grand entier non nul qui divise $\mathtt{x}$ et $\mathtt{y}$. On étend cette définition à des entiers naturels quelconques, en convenant de définir le pgcd d'un entier $\mathtt{x} \geqslant 0$ et de 0 comme valant $\mathtt{x}$.

Q26 Soit $\mathtt{f}$ une fonction de $\mathtt{E_n}$ dans $\mathtt{E_n}$, croissante au sens de l'ordre de divisibilité, et notons $\mathtt{x}_1, \ldots, \mathtt{x}_m$ les points fixes de $\mathtt{f}$ dans $\mathtt{E_n}$. Montrer que le plus petit point fixe de $\mathtt{f}$ au sens de l'ordre de divisibilité est exactement le pgcd de $\mathtt{x}_1, \ldots, \mathtt{x}_m$.

Q27 Écrire une fonction `int pgcd_points_fixes(int *t)` qui prend en argument un tableau `t` de taille $\mathtt{n}$ représentant une fonction de $\mathtt{E_n}$ dans $\mathtt{E_n}$, croissante au sens de la divisibilité, et renvoie le pgcd de ses points fixes. On impose un temps de calcul **logarithmique** en la taille $\mathtt{n}$ du tableau. On ne demande pas ici de démonstration du fait que le temps de calcul de la solution proposée est logarithmique, ceci étant le sujet de la question qui suit.

Q28 Justiifer que la fonction de la question 27 a un temps de calcul logarithmique en la taille $\mathtt{n}$ du tableau.