TP #16 — Calcul efficace d'un nombre de lapins modulo un nombre premier

Dans ce TP on s'intéresse au calcul des termes de la suite « de Fibonacci » (modulo un nombre premier, afin de se débarrasser des problèmes liés au dépassement de capacité sur le type `int` OCaml), que l'on va chercher à effectuer de façon réellement efficace. Une première approche infructueuse se base sur la fausse bonne idée d'utiliser une « forme close » sur $\mathbb{R}$ des termes de la suite (dont le calcul fait intervenir des nombres flottants). En modifiant légèrement cette approche, on aboutira à une nouvelle expression récursive des termes de la suite, qui permettra d'appliquer un algorithme d'exponentiation rapide.

On pose pour tout le TP :

```
let mod311 = 2147483647;; (* 2^31 - 1 *)
```

qui est le plus grand nombre premier dont le carré est inférieur ou égal à `max_int`, et sauf mention du contraire tout calcul sur les entiers devra se faire « modulo `mod311` ».

Calcul naïf par double récursions non indépendantes

1. Écrivez une fonction OCaml :
   ```
   fibn : int -> int
   ```
   telle que `fibn n` s'évalue en le $n$ième terme de la suite « de Fibonacci » F_n modulo `mod311`. Cette fonction **devra** utiliser une approche récursive naïve basée directement sur la définition $F_0 = 0$, $F_1 = 1$, $F_{n\geq 2} = F_{n-1} + F_{n-2}$ (où cette dernière addition se fait « modulo `mod311` »).
2. Estimez approximativement une valeur de `n` telle que le calcul `fibn n` prend environ une seconde ; environ deux secondes.

Calcul semi-naïf par registre à décalage

3. Écrivez une fonction OCaml :
   ```
   fibr : int -> int
   ```
   qui (à l'aide éventuelle d'une fonction auxiliaire) calcule F_n en utilisant un *registre* auxiliaire de deux entrées r_0 et r_1 (que vous représenterez d'une façon adaptée) et l'algorithme suivant : si $n = 0$ la fonction s'évalue en 0, sinon :
   - le registre est initialisé à $r_0 = 0$, $r_1 = 1$;
   - on procède $n - 1$ fois à une « mise à jour » du registre, en remplaçant simultanément r_0 et r_1 par r_1 et $r_0 + r_1$;

— la fonction s'évalue en la dernière valeur r_1 ainsi obtenue.

4. Exprimez le coût de votre fonction `fibr` en fonction d'un paramètre bien choisi. Ce coût est-il linéaire ou exponentiel en la taille de son argument ?
5. Prouvez la correction de l'algorithme utilisé dans `fibr`. (C'est à dire, prouvez que le résultat renvoyé par `fibr(n)` est bien F_n.)
6. Écrivez une fonction `fibcmp_nr` permettant de facilement vérifier que vos deux fonctions de calcul de F_n calculent la même chose. Utilisez-la pour tester celles-ci.
7. Estimez approximativement une valeur de `n` telle que le calcul `fibr n` prend environ une seconde ; environ deux secondes.

Fausse bonne idée : calcul par forme close dans $\mathbb{R}$

Tout ce qui suit pourrait être résumé par : allez revoir votre cours de mathématiques sur la résolution des récurrences linéaires homogènes d'ordre deux à coefficients constants (même si les outils présentés ici peuvent être différents, le principe est le même, et l'objectif identique).

L'algorithme utilisé dans `fibr` peut se formuler de la façon suivante : soit la matrice $\boldsymbol{F}$ et le vecteur $\boldsymbol{r}$ suivant :

$$\boldsymbol{F} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \qquad\qquad \boldsymbol{r} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

et pour $n \geq 1$, soit $\boldsymbol{r'} = \boldsymbol{F}^{n-1}\boldsymbol{r}$, alors `fibr n` s'évalue en $\boldsymbol{r'}_1$ (en indiçant à partir de 0).
Alternativement, on pourrait calculer $\boldsymbol{r''} := \boldsymbol{F}^n\boldsymbol{s}$ et s'évaluer en $\boldsymbol{r''}_0$. Dans le cas de `fibr` cela ajouterait une étape inutile, mais cette expression sera (légèrement) plus pratique à utiliser ci-dessous.

On peut alors *diagonaliser* $\boldsymbol{F}$ en :

$$\boldsymbol{F}_\Delta = \begin{bmatrix} \varphi & 0 \\ 0 & \overline{\varphi} \end{bmatrix}$$

c'est à dire trouver une matrice inversible $\boldsymbol{P}$ de *changement de base* telle que $\boldsymbol{P}^{-1}\boldsymbol{F}\boldsymbol{P} = \boldsymbol{F}_\Delta$

où $\varphi = \frac{1+\sqrt{5}}{2}$ et $\overline{\varphi} = \frac{1-\sqrt{5}}{2}$ sont les deux racines du polynôme $X^2 - X - 1$.

On s'autorisera par la suite à utiliser la fonction OCaml `sqrt : float -> float` de spécifications évidentes.

7. Les éléments diagonaux de l'expression ci-dessus peuvent s'obtenir par simplification de $-(\varphi+2)/(\overline{\varphi}-\varphi)$ et $(\overline{\varphi}+2)/(\overline{\varphi}-\varphi)$. Vérifiez par le calcul la validité de cette simplification :

i. « à la main »
ii. en utilisant des nombres flottants

Les puissances de $\boldsymbol{F}_\Delta$ étant triviales à calculer, on peut utiliser le changement de base de $\boldsymbol{F}$ à $\boldsymbol{F}_\Delta$ pour établir une expression « simple » pour la première coordonnée de $\boldsymbol{F}^n\boldsymbol{r}$ comme :

$$F_n = \frac{\varphi^n - \overline{\varphi}^n}{\sqrt{5}}$$

ce qui donne une « forme close » pour le terme F_n. On remarque que bien que F_n est évidemment un entier naturel, l'expression de cette forme close fait intervenir des nombres irrationnels.

8. Écrivez une fonction OCaml :

```
fibf : int -> float
```

qui utilise (des nombres flottants) pour s'évaluer en F_n *sans réduction modulo* en utilisant la forme close ci-dessus.

9. Calculez et comparez les 45 premières valeurs de (F_n) telles que calculées avec vos fonctions `fibr` et `fibf` (toutes ces valeurs étant inférieures à `mod311`, la « réduction modulo » effectuée dans `fibr` n'a ici aucun effet).

10. On donne $F_{70} = 190\,392\,490\,709\,135$ et $F_{75} = 2\,111\,485\,077\,978\,050$. Votre fonction `fibf` calcule-t'elle correctement ces termes ? Comment expliquer ce que vous observez ?

11. La fonction `fibf` peut-elle être utile pour calculer efficacement les termes de (F_n) modulo `mod311` ?

Calcul par exponentiation rapide

Par définition, φ et $\overline{\varphi}$ vérifient : $\varphi^2 = \varphi + 1$ et $\overline{\varphi}^2 = \overline{\varphi} + 1$. Il s'ensuit que n'importe quelle puissance φ^n de φ admet une expression alternative de la forme $a\varphi + b$, $a, b \in \mathbb{N}$, et de même pour $\overline{\varphi}$ (avec les *mêmes* coefficients). En calculant φ^n et $\overline{\varphi}^n$ (sur les flottants) de cette façon, on peut obtenir une expression correcte pour quelques termes de plus de (F_n), mais cela ne résout pas les problèmes plus fondamentaux liés à cette approche « flottante ». Cependant, en substituant ces nouvelles expressions pour φ^n et $\overline{\varphi}^n$ dans la forme close de F_n précédemment établie, on obtient :

$$F_n = \frac{(a\varphi + b) - (a\overline{\varphi} + b)}{\sqrt{5}}$$

ce qui se simplifie en a ! (Et par les propriétés de la matrice $\boldsymbol{P}$ diagonalisant $\boldsymbol{F}$ on obtient également que b est égal à F_{n-1}.)

Une autre façon d'exprimer ce résultat est de considérer φ comme une indéterminée, et de remarquer que l'expression $a\varphi + b$ correspond au polynôme obtenu en réduisant φ^n modulo le polynôme $\varphi^2 - \varphi - 1$, et que ces coefficients a et b correspondent respectivement à F_n et F_{n-1}. Autrement dit, $\varphi^n \equiv F_n\varphi + F_{n-1} \mod \varphi^2 - \varphi - 1$.

En utilisant ce nouveau couple d'expression pour F_n et F_{n-1} et en explicitant le calcul des coefficients a et b, on peut alors obtenir de nouvelles expressions récurrentes pratiques pour les termes de (F_n) :

$$F_{2n-1} = F_n^2 + F_{n-1}^2 \qquad F_{2n} = F_n^2 + 2F_nF_{n-1} \qquad F_{2n+1} = 2F_n^2 + 2F_nF_{n-1} + F_{n-1}^2$$

(toutes les opérations devant être faites modulo `mod311` dans notre cas).

12. Montrez la validité des nouvelles relations de récurrence ci-dessus.

13. Utilisez ces nouvelles relations pour écrire une fonction OCaml :

```
ffib_pair : int -> int * int
```

qui utilise les expressions ci-dessus et une approche par exponentiation rapide récursive telle que pour $n \geq 1$ `ffib_pair n` s'évalue en le couple des termes F_{n-1}, F_n modulo `mod311`. Cette fonction (ou une éventuelle fonction auxiliaire associée) **ne doit faire qu'un seul appel récursif**.

14. Écrivez une fonction OCaml :

```
ffib int -> int
```

qui utilise `ffib_pair` pour efficacement s'évaluer en F_n modulo `mod311`.

15. Écrivez une fonction :

```
fibcmp_fr
```

permettant de facilement vérifier que votre fonction `ffib` calcule correctement les termes de (F_n) modulo `mod311`, et utilisez-la à cette fin.

16. Exprimez le coût de votre fonction `fibr` en fonction d'un paramètre bien choisi. Ce coût est-il linéaire ou exponentiel en la taille de son argument ?

17. Estimez approximativement une valeur de `n` telle que le calcul `ffib n` prend environ une seconde ; environ deux secondes.

18. Écrivez une fonction OCaml :

```
xfib int -> int
```

qui s'évalue en F_n modulo `mod311` en utilisant le fait que F_n est l'un des coefficients du reste modulo $X^2 - X - 1$ du polynôme X^n, et que ce dernier polynôme peut se calculer efficacement par exponentiation rapide (par exemple récursivement).

Remarques. La croissance des *valeurs* des termes de (F_n) (sans « réduction modulo ») étant exponentielle en n, le coût d'un calcul « direct » de F_n par exponentiation rapide sera ultimement dominé par le coût des opérations sur des entiers de taille $|F_n|$.
L'approche par exponentiation rapide étudiée dans la dernière partie de ce TP se généralise à n'importe quelle récurrence linéaire, et cette généralisation a notamment été décrite par Fiduccia en 1985. En plus de permettre de calculer l'effectif d'une population de lapin dans un modèle à la pertinence biologique

douteuse, les récurrences linéaires interviennent couramment en calcul formel, théorie des codes & cryptographie.