DEVOIR MAISON III

Très grands nombres

Ce sujet de DM est une adaptation du sujet de TP Algo ENS 2017 « Très Grands Nombres », avec quelques questions intermédiaires en plus, et beaucoup de questions en moins pour le raccourcir. Le but est de vous introduire au « style » des TP ENS.

Les questions pratiques sont à traiter en OCaml (des fichiers contenant des fonctions pré-codées sont fournis). Les questions théoriques sont à rendre sur feuille.

Dans tout cet énoncé, on trouve des tours d'exponentiation. On fera attention à ne pas confondre a^{b^c} et $\left(a^b\right)^c$; c'est à dire que les tours d'exponentiation sont parenthésées à droite d'abord.

Dans tout cet énoncé, on notera indifféremment u_n ou $u(n)$ pour le terme d'indice n d'une suite u.

Comme toujours, n'hésitez pas à m'envoyer un mail pour me demander de l'aide !

I - Représentation trichotomique

1) Définition

Dans ce sujet, on s'intéresse à la manipulation informatique de très grands nombres, de l'ordre du gogolplex ($10^{10^{100}}$), qu'il est physiquement impossible de représenter dans le système décimal.

0. Combien de chiffres décimaux sont nécessaires pour écrire un gogoplex ?

Afin d'exploiter au mieux les capacités micro-architecturales des ordinateurs, nous étudions des variations du système de numération binaire. On note ainsi $x(p) = 2^{2^p}$.

1. Combien de bits sont nécessaires pour écrire $x(p)$? En déduire la première valeur de p pour laquelle $x(p)$ déborde. On rappelle que les entiers OCaml sont des entiers signés 63 bits.

2. Prouver que tout entier $n \geq 2$ se décompose de façon unique en un tripler $(g, p, d) \in \mathbb{N}^3$ tel que $n = g + x(p) \cdot d$ avec $0 \leq g < x(p)$ et $0 < d < x(p)$.

Partant de cette observation, nous adoptons une représentation *trichotomique* des entiers naturels sous forme d'arbres ternaires. Un noeud contient un sous-arbre gauche représentant l'entier g, un sous-arbre central représentant l'entier p et un sous-arbre droit représentant l'entier d. Par abus de langage, on écrira « g + x(p)·d » pour dénoter aussi bien un tel noeud que le nombre qu'il représente. Les feuilles permettent de représenter les nombres strictement inférieurs à $x(0) = 2$, soit 0 et 1.

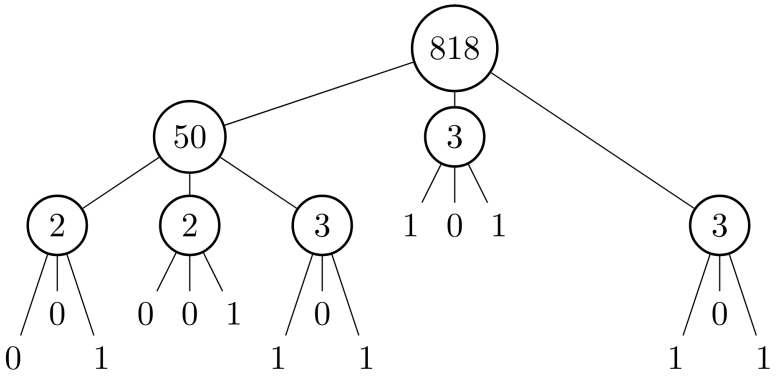

FIGURE III.1 – L'entier 818 décomposé (les étiquettes dans les noeuds sont uniquement là pour aider à la lisibilité, et ne font pas partie du type des arbres)

3. Soit $n \in \mathbb{N}$. On appelle profondeur, notée $ll(n)$, la hauteur de l'arbre ternaire représentant le nombre n dans le système trichotomique. Donner une borne supérieure asymptotique (« en grand O ») de $ll(n)$. Démontrer, par le moyen d'une suite $(a_k)_{k\in\mathbb{N}}$ à déterminer, que cette borne est asymptotiquement atteinte.

2) Deux parenthèses

2.i) Suites pseudo-aléatoires

L'énoncé d'origine, comme tous les sujets algo ENS, utilise des suites pseudo-aléatoires pour tester vos codes. C'est à dire qu'ils vous donnent un u_0, une formule de récurrence, et vous devez indiquer les valeurs trouvées par vos fonctions avec comme des entrées comme u_5, u_{100}, u_{20000}. Le u_0 étant différent selon les candidat·es, cela permet de faire des tests individuels adaptés à un concours.

Une astuce très efficace sur ces sujets consiste à *mémoïser* les suites aléatoires, c'est à dire à mémoriser tous les termes déjà calculés pour ne pas les recalculer. Nous n'avons pas encore vu comment faire ; aussi **j'ai pré-codé toutes ces suites pour vous** (cf sous-section suivante).

En bref, voici les constantes et suites utilisées par l'énoncé pour ses tests :

- $m = 2^{31} - 1$
- $(u_n)_{n\in[\![0;1000]\!]}$ définie par $u_{n+1} = (16807 \times u_n + 17) \bmod m$. La valeur u_0 vaut 42 par défaut.
- $\left(v_{k,n}\right)_{k\in[\![0;61]\!],n\in[\![0;1000]\!]}$ définie par $v_{k,n} = \left(u_n \bmod 2^k\right) + 2^k$
- $x(p)_{p\in\mathbb{N}} = 2^{2^p}$
- $h(n)_{n\in\mathbb{N}}$ définie par $h(0) = 1$ et $h(n+1) = h(n) + x_{h(n)} \times h(n)$

4. Pourquoi la limitation à $k < 62$ pour $v_{k,n}$?

5. À partir de quel n $h(n)$ provoque un débordement de capacité ? Pourquoi ?

6. Quelle est la parité de h_n ? Le prouver.

2.ii) Fichiers fournis

Le fichier `suites.ml` fournit contient une implémentation efficace des suites décrites ci-dessus. **Vous n'avez pas besoin de l'ouvrir !!** Son interface est `suites.mli` , et celle-ci contient tout ce que vous avez besoin de savoir. Si vous voulez utiliser une fonction `u` de ce fichier, appelez `Suites.u` .

Votre travail à vous est de compléter `nombres.ml` . C'est le fichier où vous allez coder les fonctions.

Il y a deux autres fichiers, `dune` et `dune-project` . Vous n'avez ni besoin de les lire ni de les comprendre. Ils servent à automatiser la compilation d'un code découpé en plusieurs fichiers sources distincts (comme c'est le cas aussi). Pour exécuter votre code, vous pouvez soit :

- Compiler avec `dune build` . Pour exécuter `dune exec ./nombres.exe`
- Lancer `utop` avec `dune utop` . Je l'ai paramétré pour que le fichier `suites.ml` soit déjà inclus : vous n'avez plus qu'à `#use "nombres.ml";;` comme d'habitude.

7. Complétez la fonction `eval : inttree -> int` qui à un arbre associe l'entier encodé. Notez qu'en pratique, on évitera de trop souvent l'utiliser car sa sortie débordera très vite : c'est justement parce que l'on veut représenter des *très* grands nombres que l'on utilise ce type d'arbres et non des entiers usuels...

8. Compléter la fonction `trouve_p : int -> int` qui prend en entrée n et calcule son entier p associé dans le triplet (g, p, d). Même remarque : en pratique, p pourra déborder et cette fonction ne servira pas trop.

9. Compléter la fonction `of_int : int -> inttree`

Remarque. Notez qu'en pratique, les trois fonctions ci-dessus ne sont pas pertinentes : on fait des arbres trichotomiques pour contourner le problème du débordement des entiers... il est donc fort probable que les sorties/entrées de ces fonctions débordent.

3) Signature

On définit la signature *Sig* d'un arbre ternaire par la formule :

$$
\begin{aligned}
Sig(0) &= 0 \\
Sig(1) &= u_{10} \bmod 97 \\
Sig(g + x(p) \times d) &= (Sig(g) + u_{20} * Sig(d)) \bmod 97 && \text{si } p \text{ est impair} \\
Sig(g + x(p) \times d) &= (Sig(g) + u_{30} * Sig(d)) \bmod 97 && \text{si } p \text{ est pair}
\end{aligned}
$$

10. Complétez la fonction `signature : inttree -> int` qui calcule la signature d'un arbre.

11. Complétez la fonction `sig_v : int -> int -> int` qui calcule $Sig(v_{k,n})$. Vous pouvez la tester avec :
- **a.** $(k, n) = (1, 10)$
- **b.** $(k, n) = (2, 20)$
- **c.** $(k, n) = (32, 30)$
- **d.** $(k, n) = (61, 40)$

12. Complétez la fonction `sig_h : int -> int` qui calcule $Sig(\ h(v_{k,7k})\ )$. Vous pouvez la tester avec :
- **a.** $k = 0$
- **b.** $k = 2$
- **c.** $k = 4$
- **d.** $k = 8$

Remarque. Pour `sig_h` , remarquez que la seule quantité qu'il est strictement nécessaire de calculer est la signature.

4) Génération

On définit un générateur pseudo-aléatoire $(gen(k, n))_{k\in[\![0;61]\!],n\in[\![0;1000]\!]}$ produisant des nombres sous forme d'arbre ternaire :

$$
\begin{aligned}
gen(0, n) &= 0 && \text{si } u_n \bmod 7 = 0\\
gen(0, n) &= 1 && \text{si } u_n \bmod 7 \neq 0\\
gen(k, n) &= g && \text{si } k > 0 \text{ et } d = 0\\
gen(k, n) &= g + x(p) \times d && \text{si } k > 0 \text{ et } d > 0\\
&\text{Où : } k' = \max(0, k - 1 - (u_n \bmod 2))\\
&\quad g = gen(k', (n + 1) \bmod 997)\\
&\quad p = v(k', n)\\
&\quad d = gen(k', (n + 2) \bmod 997)
\end{aligned}
$$

13. Complétez `sig_gen : int -> int -> int` qui calcule $Sig(gen(k, n))$. Vous pouvez la tester avec :
- **a.** $(k, n) = (6, 35)$
- **b.** $(k, n) = (8, 45)$
- **c.** $(k, n) = (10, 55)$
- **d.** $(k, n) = (12, 65)$
- **e.** $(k, n) = (14, 75)$

II - Décrémentation et incrémentation

1) Parenthèse : fonctions mutuellement récursives

En mathématiques, on peut définir deux suites mutuellement récursives. Par exemple :

$$
\begin{aligned}
u_0 &= 1 & u_{n+1} &= v_n^2 * u_n\\
v_0 &= 3 & v_{n+1} &= v_n + u_n
\end{aligned}
$$

En OCaml, on peut définir deux telles fonctions co-récursives avec `let rec u n = ... and v n = ...` :

```
let rec u n =
  if n = 0
  then 1
  else let v_n = v n in v_n * v_n * u_n

and v n =
  if n = 0
  then 3
  else v_n + u_n
```

2) Décrémentation

La décrémentation *dec*, c'est à dire l'opération qui diminue de 1 l'entier représenté, ainsi que l'opérateur $x'(p) = x(p) - 1$ sont définis correcursivement ainsi :

$$
\begin{aligned}
dec(1) &= 0 && \\
dec(g + x(p) \times d) &= dec(g) + x(p) \times d && \text{si } g \neq 0 \\
dec(g + x(p) \times d) &= x'(p) && \text{si } g = 0 \text{ et } d = 1 \\
dec(g + x(p) \times d) &= x'(p) + x(p) \times dec(d) && \text{si } g = 0 \text{ et } d \neq 1
\end{aligned}
$$

$$
\begin{aligned}
x'(0) &= 1 && \\
x'(p) &= x'(q) + x(q) + x'(q) && \text{pour } p > 0 \\
\text{Où : } q &= dec(p) &&
\end{aligned}
$$

14. Complétez `sig_dec_gen : int -> int` qui calcule $Sig(dec(gen(k, 19k \bmod 997)))$. Vous pouvez la tester avec :

- **a.** $k = 6$
- **b.** $k = 16$
- **c.** $k = 26$

3) Incrémentation

15. Décrire un algorithme réalisant l'incrémentation $inc(n) = n + 1$ dans le système trichotomique. Quelle est sa complexité en fonction de n ?

16. Calculer $Sig(inc(gen(k, 17k \bmod 997)))$. Vous pouvez la tester avec :

- **a.** $k = 6$
- **b.** $k = 7$
- **c.** $k = 16$

4) Pour aller plus loin

Le sujet d'origine est ici : `https://informatique.ens-lyon.fr/concours-info/2017/tres_grands_numbres.pdf` . Vous pouvez en faire des questions supplémentaires ; je corrigerai.

Si vous cherchez un autre sujet moins abstrait, je conseille le sujet de 2014 sur le retour sur trace (qui n'était alors pas au programme).

III - Valeurs attendues

Pour vérifier vos tests, voici les valeurs attendues avec $u_0 = 42$ (valeur par défaut de u_0) pour certaines des questions :

Q11. : a 57. b 65. c 82. d 54.

Q12. : a 65. b 11. c 4. d 88.

Q13. : a 9. b 62. c 15. d 66. e 2.

Q14. : a 63. b 93. c 37.

Q16. : a 19. b 96. c 30.

IV - Rendu

1) Rendu

⚠ Le fichier `suites.ml` a été modifié depuis la V0 de cette énoncé : téléchargez la nouvelle !!! Notez également que j'ai ajouté `nombres.mli` .

Les questions théoriques sont la partie théorique du DM, à rendre. Les fonctions dont le prototype est dans `nombres.mli` sont la partie pratique du DM, à rendre.

La partie théorique du DM est à rendre pour lundi 17 mars, avant [1] 16h00 (fin du second TD).

La partie pratique du DM est à rendre pour lundi 17 mars 2025, 20h00 (heure de Paris). Vous déposerez vos fichiers **dans un zip à votre nom** à l'adresse suivante : `https://nuage04.apps.education.fr/index.php/s/5t5CPREnAmG68sn` . Si vous avez fait ce DM à deux, mettez vos deux noms dans le titre du zip.

2) Testeur

J'ai également ajouté dans le nuage le testeur sous forme compilée (`testeur.cmi` et `testeur.cmo`), ainsi qu'un fichier `Makefile` . C'est ce que je vais utiliser pour évaluer vos codes. Dans un terminal, lancez `make` . Si votre code compile, mon testeur devrait lui donner une note (sur 20). Pour information, la partie théorique sera notée sur 10, pour un total sur 30.

1. J'enchaine les cours le lundi après-midi, donc je tiens au *avant*.