1 Problème: génération de nombres pseudo-aléatoires, une méthode peu conventionnelle...

Dans ce problème, nous allons proposer une méthode nouvelle de génération de nombres pseudo-aléatoires.
On construit une suite $(u_n)_{n\in\mathbb{N}}$ par récurrence de la manière suivante:

$$\forall n \in \mathbb{N}, u_{n+1} = \overline{\text{bin}(u_n)}^{10}$$

et avec $u_0 \in \mathbb{N}$.
Dans un soucis de simplicité, nous appelerons u_0 la *graine* de cette suite.
Par exemple, pour une graine de 4, on obtient:

$$u_0 = 4$$

$$u_1 = \overline{\text{bin}(u_0)}^{10} = \overline{100}^{10} = 100$$

$$u_2 = \overline{\text{bin}(u_1)}^{10} = 1100100$$

$$u_3 = \overline{\text{bin}(u_2)}^{10} = 100001100100101000100$$

$$u_4 = 10101101011110010110100011010110110011001101100110011001011101001 00$$

$$\vdots$$

On observe que cette suite à tendance à grandir très vite !

1.1 Questions préliminaires

On notera $|u_n|$ la nombre de chiffres dans l'écriture en base 10 u_n
Par exemple, $|4| = 1, |10020| = 5$, etc.
Attention ! $|00000| = |0| = 1$, $|010| = |10| = 2$

Q1. Donner un encadrement de $|u_n|$ en fonction de u_n. Montrer que $|u_n| \approx \log_2(u_n)$.

Afin de ne pas commettre de barbarisme mathématique, on dira que, soit $(v_n)_{n\in\mathbb{N}}$ et $(\omega_n)_{n\in\mathbb{N}}$ des suites, $v_n \approx \omega_n$, si $\frac{v_n}{\omega_n}_{n\to\infty} \to 1$

L'idée est d'utiliser des logarithmes pour encadrer les valeurs. En effet, on a:

$$\forall n \in \mathbb{N}, \log_2(n) + 1 \geq |\text{bin}(n)| \geq \log_2(n)$$

Ceci donne, avec des entiers naturels,

$$\forall n \in \mathbb{N}, \log_2(u_n) + 1 \geq |u_n| \geq \log_2(u_n)$$

On observe donc naturellement que:

$$\forall n \in \mathbb{N}, \frac{\log_2(u_n)+1}{\log_2(u_n)} \geq \frac{|u_n|}{\log_2(u_n)} \geq \frac{\log_2(u_n)}{\log_2(u_n)}$$

$$\forall n \in \mathbb{N}, 1+\frac{1}{\log_2(u_n)} \geq \frac{|u_n|}{\log_2(u_n)} \geq 1$$

Par théorème d'encadrement, on en déduit que

$$\frac{|u_n|}{\log_2(u_n)}_{n\to\infty} \to 1$$

D'où $|u_n| \approx \log_2(u_n)$
En réalité, il s'agit de bien plus qu'un équivalent, puisque notre inéquation nous montre qu'il est exacte à ± 1, ce qui est extrêmement précis. C'est pour cette raison qu'on s'autorisera la notation extrêmement permissive $\approx$, qui nous autorisera à faire des compositions à gauche.

Q2. Etudier la limite de la suite u_n en fonction de la graine

Lorsque $a > 1$, montrer que la suite est strictement croissante et à valeur dans N est suffisant, puisque celà implique nécessairement que la suite tend vers ∞
Montrer que la suite u_n est à valeur dans $\mathbb{N}$ est trivial ; il suffit de dire que l'on ne travaille qu'avec des entiers.
Justifier de sa stricte croissance est plus subtil. On peut se baser sur l'étude de la taille des nombres, (voir q1), car une nombre infiniement long (dans $\mathbb{N}$) est a priori ∞.

Q3. On observe/admet que

$$\forall n \in \mathbb{N}, \quad u_{n+1} \approx 10^{|u_n|}$$

Montrer que:

$$\forall u_0 \in \mathbb{N}_{\geq 2}, \forall n \in \mathbb{N}, \quad |u_n| \approx u_0 \log_2(10)^n$$

Soit $a \in \mathbb{N}_{\geq \not\vdash}$ la graine.

$$\forall n \in \mathbb{N}, |u_{n+1}| \approx \log_2(u_{n+1})$$

$$\forall n \in \mathbb{N}, |u_{n+1}| \approx \log_2(10^{|u_n|})$$

$$\forall n \in \mathbb{N}, |u_{n+1}| \approx |u_n| \log_2(10)$$

On remarque qu'il s'agit d'une suite géométrique, d'où

$$\forall n \in \mathbb{N}, |u_n| \approx |u_0| \log_2(10)^n$$

Q4. Expliquer comment la croissance exponentielle de $|u_n|$ entraine une limite à l'utilisation de cette méthode en C

Elle va très vite (très très vite même) entrainer un dépassement d'entier, et donc un UB en C

1.2 Implémentation

Afin de résoudre ce problème, nous allons "sectionner" le nombre obtenu, en ne gardant que les $\delta \in \mathbb{N}$ premiers chiffres à chaque étape, en commençant par le deuxième. On appelera ce nombre *l'indice de sectionnement* de l'algorithme.
Par exemple, pour $u_n = 10100010100101001$ et $\delta = 4$, on gardera seulement 0100

On appelera δ-compatible tout entier qui peut s'écrire à l'aide de δ chiffres et ne comportant que des 0 ou des 1
Par exemple, pour $\delta = 5$, 10100 ou 10 sont δ-compatible. En revanche, 1020 ou 100001000100010 ne le sont pas.

Question bonus: Pourquoi commence-t-on par le deuxième caractère et non pas le premier ?

Le premier caractère est pas très aléatoire... il s'agit toujours d'un '1'

Q5. Trouver le $\delta \in \mathbb{N}$ qui maximise le nombre obtenu (afin d'avoir un aléatoire de "bonne qualité") tout en prenant en compte la limite exposée Q4

Rappel: $2^{31} - 1 = 2147483647$ *(cette valeur est à apprendre par coeur !)*

La réelle contrainte ici est de ne jamais dépasser $2^{31} - 1$, limite des entiers 32 bits signés en C.
Les nombres que nous allons manipuler seront toujours composés de 0 et de 1. Ainsi, le nombre à ne pas dépasser est en réalité:
1111111111 (ce qui correspond à $\text{len}(2^{31} - 1) = 10$ fois le chiffre '1')
Ainsi, en choisissant $\delta = 10$, on s'assure que les nombres manipulés ne pourront jamais dépasser la limite de la mémoire qui leur est allouée (sauf magie noir hors programme).

Q6. Ecrire une fonction bin prenant en argument un entier, supposé positif, et renvoyant une chaine de caractère représentant sa décomposition binaire. En donner la complexité.

```
char* bin(int number) {
    if(number == 0) {
        char* result = malloc(2);
        result[0] = '0';
        result[1] = '\0';
        return result;
    }

    int size = (int) log2(number);
    char* result = malloc(sizeof(char) * size + 1);

    for(int i = size; i>=0; i--) {
        result[i] = (number % 2) ? '1' : '0';
        number /= 2;
    }

    result[size + 1] = '\0';

    return result;
}
```

Evidemment, avant de parler de complexité, il faut prouver la terminaison ! Elle est ici triviale avec un invariant de structure size - i, qui est clairement à valeur dans $\mathbb{N}$ et strictement décroissant. Outre cela, l'invariant de structure de math.h nous donne la terminaison de log2, donc pas de soucis particuliers.
La complexité est ici $\theta(\log_2(n)) = \theta(\ln(n))$

Q7. Ecrire une fonction extract prenant en argument une chaine de caractère et un entier δ, et renvoyant les δ premiers caractères à partir du deuxième de la chaine passée en argument. Si la chaine n'est pas assez grande, retourner la chaine sans modifications. En donner la complexité et prouver sa correction.

à finir

Q8. Ecrire une fonction aleatoire, prenant en argument un entier α, un entier δ et une entier $\aleph$ et renvoyant l'entier généré "aléatoirement" par notre méthode avec pour graine α et pour indice de sectionnement δ, en l'itérant $\aleph$ fois (c'est à dire, (aleatoire$(\alpha, \delta))^{\aleph}$ au sens de $\circ$).

Explication alternative de $\aleph$:

$$\text{aleatoire}(\alpha, \delta, \aleph) = \text{aleatoire}(\text{aleatoire}(\cdots \text{aleatoire}(\alpha, \delta, 1), \delta, 1))$$

ou bien:

$$\text{aleatoire}(\alpha, \delta, \aleph) = \text{aleatoire}(\text{aleatoire}(\alpha, \delta, 1), \delta, \aleph - 1))$$

à finir

1.3 Vérifications

Afin d'évaluer la qualité des algorithmes de génération de pseudo-aléatoire, il peut être utile d'étudier les cycles qui existent au sein de ceux-ci.
On peut démontrer par la méthode des tiroirs et des chaussettes qu'il existe au moins un cycle de taille maximale. En réalité, un tel cycle est presque impossible à atteindre, et l'algorithme produira plutôt beaucoup de petits cycles qui ne sont pas reliés. On peut assimiler cela à la notion de spé d'**attracteur** et de **piège** (qui ne sera très rapidement abordée ici).

On appelle **cycle** toute famille $(x_0, \cdots, x_n)$ d'entiers δ-compatibles tels que

$$\forall i \in [|0; n-1|], \quad \text{aleatoire}(x_i, \delta, 1) = x_{i+1}$$

et $x_0 = x_n$

Par exemple, si on a:

$$\begin{aligned} \text{aleatoire}(4, \delta, 1) &= 1010 \\ \text{aleatoire}(1010, \delta, 1) &= 1110 \\ \text{aleatoire}(1110, \delta, 1) &= 1000 \\ \text{aleatoire}(1000, \delta, 1) &= 1010 \end{aligned}$$

Alors $(1010, 1110, 1000)$ forme un cycle.

On appelle **piège** toute famille $x_0 \cdots x_n$ d'entiers δ-compatibles tels que

$$\forall \aleph \in \mathbb{N}, \forall i \in [|0; n|], \exists j \in [|0; n|] \quad \text{aleatoire}(x_i, \delta, \aleph) = x_j$$

On note Pr l'ensemble des entiers δ-compatibles qui appartiennent à un piège.

On appelle **attracteur** tout entier x δ-compatible tels que

$$\exists n \in \mathbb{N}, \quad \text{aleatoire}(x, \delta, n) \quad \text{appartienne à un piège}$$

On note Atr l'ensemble des entiers δ-compatibles qui sont attracteurs.

On observe que tout entier δ-compatible appartenant à un piège est un attracteur.
On définit alors la notion **d'attracteur stricte** comme étant tout attracteur qui n'appartient pas à un piège.

On note AtrS l'ensemble des entiers δ-compatibles qui sont des attracteurs strictes.

Q9. Montrer que pour toute famille d'entiers δ-compatibles $(x_0 \cdots x_n)$,

$$(x_0 \cdots x_n) \textbf{ est un cycle} \iff (x_0 \cdots x_n) \textbf{ est un piège}$$

Soit $n \in \mathbb{N}$, $(x_0 \cdots x_n)$ une famille d'entiers δ-compatibles. Procédons par double-implication:
$\Rightarrow$ Supposons que $(x_0 \cdots x_n)$ soit un cycle. Soit $i \in [|0; n|]$. Si $0 \leq \aleph - i < n$, alors on a aleatoire$(x_i, \delta, \aleph) = x_{\aleph - i} \in (\mathrm{x}_0 \cdots x_n)$. Autrement, aleatoire$(x_i, \delta, \aleph) = x_{(\aleph - i) \mod n)} \in (\mathrm{x}_0 \cdots x_n)$
$\Leftarrow$ Supposons que $(x_0 \cdots x_n)$ soit un piège. Alors, en particulier, pour $\aleph = 1$, la propriété reste vraie. Il suffit de réorganiser les éléments de $(x_0 \cdots x_n)$ en fonction de leurs successeurs respectifs.

Q10. Montrer que pour tout entiers δ-compatibles x,

$$x \textbf{ n'appartient pas à un un cycle} \Longrightarrow x \textbf{ est un attracteur stricte}$$

En déduire que

$$\mathbf{Pr} \bigsqcup \mathbf{AtrS} = \{x \in \mathbb{N} | x \quad \delta\textbf{-compatible}\} = \mathbb{N}_\delta \quad \textbf{(notation)}$$

Soit x un entier δ-compatible n'appartenant pas à un cycle. Supposons par l'absure que x ne soit pas un attracteur stricte. On observe que l'ensemble des entiers δ-compatibles est une partie de $\mathbb{N}_{\leq 10^{\delta+1}}$, donc est fini. x n'appartient à aucun cycle, à chaque itération d'aleatoire sur x, le nombre obtenu est nouveau, et est δ-compatible. Ainsi, en appliquant $10^{\delta+1}$ fois l'algorithme, on obtient $10^{\delta+1} + 1 \leq \mathrm{Card}(\mathbb{N}_{\leq 10^{\delta+1}})$ ce qui est absurde.

Q11. Montrer que pour tout $x \in \mathbb{N}_\delta \setminus \{0; 1\}$,

$$|x| < \delta \Longrightarrow x \textbf{ est un attracteur stricte.}$$

Indication: la stricte croissance d'u_n pour une graine > 2 peut servir...

Soit $x \in \mathbb{N}_\delta \setminus \{0; 1\}$,
on a déja montré que $\mathrm{bin}(x) > x$ (car $x \neq 0$ et $x \neq 1$). Ainsi, si $|x| < \delta$, on distingue deux cas:
1er cas: $|\text{aleatoire}(x, \delta, 1)| < \delta$: alors on a pas d'extraction, donc la stricte croissance établie avant nous dit qu'il est impossible de revenir sur x. Donc il n'y a pas de cycle.
2ème cas: $|\text{aleatoire}(x, \delta, 1)| = \delta$: alors on restera toujours dans de l'extraction, et donc de taille δ. Il est donc impossible de revenir sur x, donc il n'y a pas de cycle.

Q12. Donner une majoration de Card(Pr) en fonction de Card$(\mathbb{N}_\delta)$ et δ

$\mathrm{Pr} \bigsqcup \mathrm{AtrS} = \mathbb{N}_\delta$ donc $|\mathrm{Pr} \bigsqcup \mathrm{AtrS}| = |\mathrm{Pr}| + |\mathrm{ArtS}| = |\mathbb{N}_\delta|$ càd $|\mathrm{Pr}| = |\mathbb{N}_\delta| - |\mathrm{ArtS}|$ d'où
$|\mathrm{Pr}| \leq |\mathbb{N}_\delta| - |\mathbb{N}_{\delta-1}| + 2$

Q13. Ecrire une fonction cycle, prenant en argument un entier δ-compatible, supposé positif, et renvoyant la taille du cycle que forme cet entier par l'algorithme de notre "nouvelle méthode"

Q14. Ecrire une fonction max, renvoyant la graine maximisant la taille du cycle pour des nombres δ-compatibles