LE CRIBLE D'ÉRATHOSTÈNE

Raymond SEROUL

1. Description de l'algorithme

Pour déterminer tous les nombres premiers compris entre 2 et N, on pratique, dans les petites classes, un algorithme connu sous le nom de *crible d'Érathostène*. Rappelons en quoi consiste cet algorithme. On écrit tous les entiers de l'intervalle $[2\,..\,N]$, puis on barre les multiples stricts de 2 (ici $N = 50$) :

		2	3	.	5	.	7	.	9	.
	11	.	13	.	15	.	17	.	19	.
(T_2)	21	.	23	.	25	.	27	.	29	.
	31	.	33	.	35	.	37	.	39	.
	41	.	43	.	45	.	47	.	49	.

Le premier entier non barré après 2 étant 3, on barre les multiples de 3 :

		2	3	.	5	.	7	.	.	.
	11	.	13	.	.	.	17	.	19	.
(T_3)	.	.	23	.	25	.	.	.	29	.
	31	.	.	.	35	.	37	.	.	.
	41	.	43	.	.	.	47	.	49	.

Le premier entier non barré après 3 étant 5, on barre les multiples de 5 :

		2	3	.	5	.	7	.	.	.
	11	.	13	.	.	.	17	.	19	.
(T_4)	.	.	23	.	.	.	.	.	29	.
	31	.	.	.	.	.	37	.	.	.
	41	.	43	.	.	.	47	.	49	.

Le premier entier non barré après 5 étant 7, on barre les multiples de 7 :

		2	3	.	5	.	7	.	.	.
	11	.	13	.	.	.	17	.	19	.
(T_5)	.	.	23	.	.	.	.	.	29	.
	31	.	.	.	.	.	37	.	.	.
	41	.	43	.	.	.	47	.	.	.

Le processus s'arrête ici car il n'y a plus rien à barrer. Les entiers qui restent sont les nombres premiers recherchés.

2. Le crible d'Érathostène (version classique)

3. Mise en forme de l'algorithme. — Nous obtiendrons très naturellement un programme si nous traduisons les opérations précédentes à l'aide d'une ou plusieurs suites récurrentes. Une première suite est naturellement :

$$T_t = \text{ensemble des entiers non barrés à l'instant } t.$$

Par convention, $T_1 = [2\,..\,N]$. Les ensembles $T_2, \ldots, T_5$ sont ceux du paragraphe précédent.

Mais cette suite ne suffit pas; pour déduire T_{t+1} de T_t, nous avons besoin d'une information supplémentaire : la suite (p_t) des "premiers non barrés" :

$$p_{t+1} = \min\{n \in T_t \mid n > p_t\}, \quad p_1 = 2.$$

On a donc $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, $p_4 = 7$, $p_5 = 11$ *etc.*

En désignant par Mult_Stricts(p) l'ensemble des multiples stricts de p (c'est-à-dire l'ensemble des kp avec $k > 1$), nous avons :

$$T_{t+1} = T_t \setminus \text{Mult}(p_t).$$

Une première description algorithmique du crible d'Érathostène pourrait être :

$t := 1$; $T_1 := [2\,..\,N]$; $p_1 := 2$;
while $T_t \cap]p_t, N] \neq \emptyset$ **do begin**
  $T_{t+1} = T_t \setminus \text{Mult\_Stricts}(p_t)$;
  $p_{t+1} = \min\{T_{t+1} \cap]p_t, N]\}$;
  $t := t + 1$
end

Crible d'Érathostène (version incorrecte).

Le test de sortie de boucle $T_t \cap]p_t, N] \neq \emptyset$ est une tentative de traduction de l'énoncé imprécis : "on barre tant que c'est possible".

Présenté ainsi, cet algorithme présente deux défauts :

- Le test de sortie de la boucle est particulièrement déplaisant :

$$T_t \cap]p_t, N] \neq \emptyset.$$

Devrons-nous parcourir l'intervalle $]p_t, N]$ pour y rechercher un élément de T_t ? Ce n'est pas très efficace! D'autre part, il n'est pas possible de remplacer ce test par la condition $p_t < N$, car $N = 28$ et $p_8 = 23$ fournissent un contre-exemple.

• Plus grave encore, cet algorithme se termine immanquablement par un plantage! Lorsque p_t est le plus grand nombre premier $\leq N$, l'algorithme — qui n'est pas au courant — continue en essayant de trouver le plus petit élément de l'ensemble vide $T_{t+1} \cap]p_t, N]\ldots$

Nous allons améliorer considérablement le test de sortie de boucle en nous inspirant d'un résultat bien connu : un entier p est premier s'il n'est divisible par aucun entier de l'intervalle $2 \leq d \leq \sqrt{p}$.

Avant de continuer, rappelons un résultat — dont la démonstration est assez technique[1] — de la théorie des nombres, résultat connu sous le nom de *postulat de Bertand.* Cet énoncé (qui n'a été que conjecturé par BERTRAND en 1845) a été démontré par TCHEBYCHEFF dès 1850 :

THÉORÈME (postulat de Bertrand). — *Soit (p_t) la suite des nombres premiers : $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, etc. Alors, pour tout $t \geq 1$, on a*

$$p_{t+1} < 2p_t.$$

Sachant que pour $n \geq 2$ on a $2n \leq n^2$, on obtient tout de suite le :

COROLLAIRE. — *Soit p un nombre premier. On peut toujours trouver un nombre premier q qui satisfait les inégalités :*

$$p < q < p^2.$$

Nous pouvons maintenant améliorer le test d'arrêt :

$t := 1$; $T_1 := [2\,..\,N]$; $p_1 := 2$;
while $p_t^2 \leq N$ **do begin**
  $T_{t+1} = T_t \setminus \text{Mult\_Stricts}(p_t)$;
  $p_{t+1} = \min\{T_{t+1} \cap]p_t, N]\}$;
  $t := t + 1$
end

Crible d'Érathostène (version correcte).

Démonstration. — Pour démontrer que cet algorithme est exact, nous devons nous assurer :

• que l'algorithme ne va pas boucler indéfiniment (i.e. que l'on sort toujours de la boucle while au bout d'un temps fini);

[1] Une démonstration élémentaire figure dans *An Introduction to the Theory of Numers*, Hardy and Wright, pages 343–344.

• que l'algorithme ne plante pas (i.e. que p_{t+1} est bien défini ou, ce qui revient au même, que l'ensemble $T_{t+1} \cap]p_t, N]$ n'est jamais vide);

• que le dernier ensemble T_t obtenu contient tous les nombres premiers $\leq N$ et rien d'autre.

Considérons l'hypothèse de récurrence :

$$(\mathcal{H}_t) \quad \begin{cases} \text{(i) tous les nombres premiers} \leq N \text{ appartiennent à } T_t ; \\ \text{(ii) les } t \text{ premiers nombres premiers sont } p_1, \ldots, p_t ; \\ \text{(iii) } T_t \text{ ne contient aucun multiple strict des nombres} \\ p_1, \ldots, p_{t-1}. \end{cases}$$

Montrons que l'hypothèse $(\mathcal{H}_t)$ est vraie *chaque fois que l'on se présente à l'entrée de la boucle* (les informaticiens appellent $(\mathcal{H})$ un *invariant* de la boucle).

Il est clair que $(\mathcal{H}_1)$ est vérifiée. Supposons alors $(\mathcal{H}_t)$ vraie à l'entrée dans la boucle et supposons $p_t^2 \leq N$, ce qui nous permet d'entrer de nouveau dans celle-ci.

Nous savons que p_t appartient à T_t d'après $(\text{i})_t$ et $(\text{ii})_t$. Soit q le premier nombre premier que l'on rencontre après p_t. Le corollaire du postulat de Bertrand nous apprend que l'on a $p_t < q < p_t^2$. La condition $(\text{i})_t$ montre alors que q appartient à T_t, d'où il résulte que $T_{t+1} \cap]p_t, N]$ n'est jamais l'ensemble vide. Par conséquent, p_{t+1} existe à chaque passage dans la boucle.

Prouvons alors que $(\mathcal{H}_{t+1})$ est vérifiée lorsque l'on sort de la boucle et que l'on se présente de nouveau à l'entrée :

• L'ensemble T_{t+1} est obtenu en ne supprimant dans T_t que les multiples tricts de p_t : les conditions $(\text{i})_{t+1}$ et $(\text{iii})_{t+1}$ sont satisfaites.

• Supposons que $(\text{ii})_{t+1}$ soit fausse, i.e. que p_{t+1} ne soit pas le plus petit nombre premier $> p_t$. Avec les notations précédentes, on aurait $p_t < q < p_{t+1}$, ce qui contredit la définition de p_{t+1}.

Nous savons maintenant que l'algorithme ne plante jamais. Comme la suite des nombres p_t est strictement croissante et qu'elle est majorée par N, cela nous apprend que l'algorithme ne peut pas boucler indéfiniment.

Il reste une dernière formalité : montrer que l'ensemble T_t, au moment de l'arrêt, contient tous les nombres premiers $\leq N$ et rien d'autre. La condition $(\text{i})_t$ montre que tous les nombres premiers $\leq N$ appartiennent à T_t. Supposons que T_t contient un entier n composé. Le plus petit diviseur premier q de n vérifie $n = qn'$ et $q^2 \leq n \leq N$. Les conditions $(\text{ii})_t$ et $(\text{iii})_t$ nous apprennent que $q > p_{t-1}$. De $(\text{ii})_t$, on tire $q \geq p_t$. Comme nous nous plaçons à la sortie de la boucle, nous savons que $p_t^2 > N$, d'où la contradiction $q^2 \geq p_t^2 > N$. □

4. Passage au programme

La description de l'algorithme est une description mathématique, ce qui inclut la gestion de l'indice t (l'instruction $t := t + 1$). Pour obtenir un "vrai" algorithme (c'est-à-dire un programme), il suffit de se débarrasser du temps t. Pour cela, on utilise la correspondance suivante : on interprète x_t comme le *contenu* de la mémoire x à l'instant t. De cette manière, la relation de récurrence $x_{t+1} = f(x_t)$ se trandforme en l'affectation $x := f(x)$ et nous obtenons le programme :

$T := [2\,..\,N]$; $p := 2$;
while $p^2 \leq N$ **do begin**
  $T := T \setminus \text{Mult\_Stricts}(p)$;
  $p := \min\{T \cap]p, N]\}$;
end

Crible d'Érathostène (version classique).

Mais cette description est encore trop mathématique; la représentation des ensembles T_t dans la machine n'est pas précisée. Une solution naturelle consiste à choisir un tableau de booléens que nous appellerons est_barre :

$$\text{est\_barre}[n] = \begin{cases} \textit{true} & \text{si } n \text{ est barré,} \\ \textit{false} & \text{sinon.} \end{cases}$$

Nous choisirons aussi de travailler sur place, c'est-à-dire dans le même tableau. L'ensemble T_t est donc l'*état* du tableau est_barre à l'instant t.

Les déclarations PASCAL sont donc :

```
const max = 50 ;
type tableau = array[2 .. max] of boolean ;
var est_barre : tableau ; p : integer ;
```

Nous supposerons que nous disposons :

- de la procédure barrer_mult_stricts(p, T) qui barre, dans le tableau T, les multiples stricts de p;
- de la fonction premier_non_barre(p, T) qui retourne le plus petit entier $q \in T$ vérifiant la condition $q > p$.

Ceci précisé, le corps principal du programme PASCAL est :

```
begin
  for p := 2 to max do est_barre[p] := false ; p := 2 ;
  while p * p <= max do begin
      barrer_mult_stricts(p, est_barre) ;
      p := premier_non_barre(p, est_barre)
    end
end.
```

Pour barrer les multiples stricts de p, il est préférable d'utiliser des additions répétées ($x := p + p$ et $x := x + p$), car une addition est au moins dix fois plus rapide qu'une multiplication :

```
procedure barrer_mult_stricts(p : integer ; var est_barre : tableau) ;
var x : integer ;
begin
| x := p + p ;
| while x <= max do begin est_barre[x] := true ; x := x + p end ;
end ;
```

Écrire la fonction premier_non_barre ne présente pas de difficulté particulière. Remarquez simplement la déclaration 'var' qui évite une recopie inutile en mémoire du tableau est_barre (cela ne présente aucun danger puisque l'on ne modifie pas le tableau) :

```
function premier_non_barre(var est_barre : tableau) : integer ;
var x : integer ;
begin
| x := p + 1 ;
| while est_barre[x] do x := x + 1 ;
| premier_non_barre := x
end ;
```

Remarque. — On peut améliorer les performances de ce programme :

- Il est maladroit de partir de l'intervalle $[2\,..\,N]$ pour supprimer tous les entiers pairs à l'instruction suivante.
- On peut remplacer les appels

barrer_mult_stricts$(p, \text{est_barre})$ er premier_non_barre$(p, \text{est_barre})$

par les codes correspondants. Cela évite au programme de perdre du temps lorsqu'il se branche sur le code de la procédure ou de la fonction.

- Les multiples stricts de p sont $2p$, $3p$, $4p$, $5p$, $6p\ldots$ Si p est impair, nous savons que les multiples pairs de p ont déjà été barrés. On accélère notablement le processus en ne barrant que les multiples $3p$, $5p$, $7p\ldots$, ce qui s'obtient en répétant l'instruction $q := q + r$, avec $r := p + p$.

Ces transformations fournissent un algorithme rapide, mais plus difficile à comprendre (voir figure).

```
T := [2..N] \ {4,6,8,...} ; p := 3 ;
while p² ≤ N do begin
    r := p + p ; q := p + r ;
    while q ≤ N do begin T := T \ {q} ; q := q + r end ;
    p := p + 2 ; while p ∉ T do p := p + 2
end
```

Crible d'Érathostène (version rapide et illisible).

5. Un algorithme très intelligent

6. Critique de l'algorithme classique. — L'algorithme que nous venons de développer est inefficace car un entier composé est barré autant de fois qu'il possède de diviseurs premiers distincts. Puisque 30 est divisible par 2, 3 et 5, il est barré trois fois. Le travail superflu n'est pas négligeable :

• Pour $N = 100$, l'algorithme barre 113 entiers et trouve 25 nombres premiers : il effectue donc $113 - (99 - 25) = 39$ suppressions inutiles.

• Pour $N = 1\,000$, l'algorithme barre 1 549 entiers et trouve 168 nombres premiers, ce qui fait 718 suppressions inutiles.

• Et la situation s'aggrave lorsque N augmente, car les nombres premiers se raréfient. Ainsi, pour $N = 10\,000$, il y a 9 221 suppressions inutiles et pour $N = 100\,000$, cela passe à 111 747 !

Il est donc très tentant de rechercher un algorithme qui ne barrerait qu'une seule fois chaque entier composé. Mais, lorsqu'on supprime les multiples stricts de 3, comment savoir qu'il est inutile de barrer 6 alors qu'il faut barrer 9 ? La réponse consiste à calculer le *plus petit diviseur* de ces deux nombres : $\mathrm{ppd}(6) = 2$ et $\mathrm{ppd}(9) = 3$ (le ppd est un nombre premier). Cette remarque nous amène à classifier les nombres composés à l'aide de leur ppd :

PROPOSITION. — *Soient $n > 1$ un entier composé et $p = \mathrm{ppd}(n)$ son plus petit diviseur. Alors n s'écrit de manière unique $n = p^\ell m$ si ℓ et m vérifient les conditions:*

$$\text{(C)} \qquad \ell \geq 1, \quad m = p \text{ ou } \mathrm{ppd}(m) > p.$$

Démonstration. — Écrivons $n = p^\ell m$ avec $\ell \geq 1$ et p ne divisant pas m. Deux cas se présentent :

• Si $m = 1$, on a $n = p^\ell$ avec $\ell \geq 2$ puisque n est composé. L'écriture cherchée est alors $n = p^{\ell-1} \times p$.

• Si $m > 1$, l'écriture cherchée est $n = p^\ell \times m$.

L'unicité de cette écriture est évidente. □

NOTATION. — Nous utiliserons l'écriture $((p, m, \ell))$ pour désigner un triplet (p, m, ℓ) qui vérifie la condition (C).

7. Un ordre nouveau. — Soit $N > 2$ un nombre entier et posons :

$$S_N = \textit{ensemble des entiers composés } n \textit{ tels que } 2 \leq n \leq N.$$

Nous identifierons un entier composé $n \in S_N$ avec le triplet $((p, m, \ell))$ qui lui est associé; autrement dit, nous pratiquerons l'abus d'écriture $n = ((p, m, \ell))$.

Notons $\lessdot$ l'ordre lexicographique sur $\mathbb{N}^3$:

$$(a, b, c) \lessdot (u, v, w) \quad \text{si} \quad (a < u) \text{ ou } (a = u \text{ et } b < v) \text{ ou } (a = u \text{ et } b = v \text{ et } c < w).$$

Il s'agit d'un ordre total.

Puisque nous identifions les éléments de S_N avec des triplets, nous pouvons munir S_N de l'ordre lexicographique, ce qui fournit un ordre total que nous notons encore $\lessdot$.

Exemple. — Les éléments de S_{21}, décomposé en triplets, sont :

$4 = ((2, 2, 1))$	$10 = ((2, 5, 1))$	$16 = ((2, 2, 3))$
$6 = ((2, 3, 1))$	$12 = ((2, 3, 2))$	$18 = ((2, 9, 1))$
$8 = ((2, 2, 2))$	$14 = ((2, 7, 1))$	$20 = ((2, 5, 2))$
$9 = ((3, 3, 1))$	$15 = ((3, 5, 1))$	$21 = ((3, 7, 1))$.

L'ordre total $\lessdot$ sur l'ensemble S_{21} est donc :

$$4 \lessdot 8 \lessdot 16 \lessdot 6 \lessdot 12 \lessdot 10 \lessdot 20 \lessdot 14 \lessdot 18 \lessdot 9 \lessdot 15 \lessdot 21.$$

On voit d'abord apparaître les multiples de 2, puis les multiples de 3. En effet, dans l'ordre lexicographique, la première coodonnée est la 'plus importante'; autrement dit, on a l'implication :

$$\text{ppd}(n) < \text{ppd}(n') \implies n \lessdot n'.$$

8. La fonction successeur dans S_N

Le charme de l'ordre lexicographique est qu'il se programme très simplement. Si $E = E_1 \times E_2 \times E_3$ est un ensemble de triplets (u, v, w), on parcourt linéairement E dans l'ordre lexicographique croissant à l'aide de trois boucles emboîtées :

for $u \in E_1$ **do**
  for $v \in E_2$ **do**
    for $w \in E_3$ **do** ...

Pour parcourir l'ensemble S_N, nous utiliserons donc trois boucles emboîtées :

for $p := 2, 3, 5, 7, \ldots$ **do**
  for $m := m_1, m_2, \ldots$ **do**
    for $\ell := 1, 2, 3, 4, \ldots$ **do** ...

Mais une difficulté surgit : S_N n'est pas un produit de trois intervalles; on a seulement une inclusion $S_N \subset E_1 \times E_2 \times E_3$. Comme les variables p, m et ℓ n'appartiennent pas à un intervalle fixe, le calcul des valeurs successives de p, m et ℓ se complique passablement :

THÉORÈME. — *Soient* $n = ((p, m, \ell)) \in S_N$ *un entier composé* $\leq N$ *et* T *l'intervalle* $[2\,..\,N]$ *privé des entiers composés* $\leq n$. *On munit* S_N *de l'ordre lexicographique et* T *de l'ordre ordinaire; ces deux ordres définissent des fonctions successeur que nous noterons* succ(, S_N) *et* succ(, T). *Dans ces conditions, la deuxième composante* m *de* n *appartient à* T *et l'on a :*

(i) *Si* $pn \leq N$, *on a* $\mathrm{succ}(n, S_N) = pn$.

(ii) *Si* $pn > N$, *alors* $m' = \mathrm{succ}(m, T)$ *existe.*

(iii) *Si* $pn > N$ *et si* $pm' \leq N$, *on a* $\mathrm{succ}(n, S_N) = pm'$.

(iv) *Si* $pn > N$ *et si* $pm' > N$, *l'ensemble* T *ne contient aucun multiple de* p *et* m' *est un nombre premier que nous noterons* p'.

(v) *Si* $pn > N$, *si* $pm' > N$ *et si* $p'^2 \leq N$, *on a* $p'^2 = \mathrm{succ}(n, S_N)$.

(vi) *Si* $pn > N$, *si* $pm' > N$ *et si* $p'^2 > N$, *l'entier* n *n'a pas de successeur dans* S_N *et* T *est l'ensemble de tous les nombres premiers* $\leq N$.

Cet énoncé étant très technique, familiarisons-nous d'abord avec celui-ci à l'aide de l'ensemble S_{50}.

La partie (i) de l'énoncé est la plus intuitive; pour obtenir les successeurs des entiers 4, 8, 16, 6, 12, 24, 10, 20, 14, 18, 22, 9 et 15, on multiplie chacun de ces entiers par son ppd.

La partie (iii) s'applique quand le successeur de n a même ppd que n mais lorsque ℓ a une valeur maximum; il faut alors changer de valeur de m. C'est le cas lorsque $n = 32$ puisque $32 = ((2, 2, 4))$ et $2 \times 32 > 50$. Dans cette situation, on a $T = \{2, 3, 5, \ldots\}$. Par conséquent, le successeur de 32 dans S_{50} est $2 \times \mathrm{succ}(2, T) = 2 \times 3 = 6$.

$n = p^\ell m$	p	m	ℓ	$n = p^\ell m$	p	m	ℓ
4, 8, 16, 32	2	2	1, 2, 3, 4	46	2	23	1
6, 12, 24, 48	2	3	1, 2, 3, 4	50	2	25	1
10, 20, 40	2	5	1, 2, 3	9, 27	3	3	1, 2
14, 28	2	7	1, 2	15, 45	3	5	1
18, 36	2	9	1, 2	21	3	7	1
22, 44	2	11	1, 2	33	3	11	1
26	2	13	1	39	3	13	1
30	2	15	1	25	5	5	1
34	2	17	1	35	5	7	1
38	2	19	1	49	7	7	1.
42	2	21	1				

L'ensemble S_{50} ordonné lexicographiquement.

La partie (v) intervient lorsque l'on a épuisé tous les entiers composés ayant un même ppd. Par exemple, le dernier entier pair de S_{50} est $n = 50$; on a $50 = ((2, 25, 1))$, $p = 2$, $m = 25$ et

$$T = \{2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, \ldots\}.$$

Les successeurs dans T de $m = 25$ et de $p = 2$ sont respectivement $m' = 27$ et $p' = 3$. Comme pm' est trop grand, le successeur de 50 est donc $p'^2 = 9$.

9. Un algorithme sophistiqué. — La détermination explicite du successeur dans l'ensemble S_N suggère un algorithme très intelligent, qui barre une fois et une seule chaque entiers composé de l'intervalle $[2\,..\,N]$. Nous reconnaissons les trois boucles emboîtées qui permettent le parcours linéaire de S_N et les tests qui permettent de sortir de chaque boucle.

Le Théorème de la partie 3.3 montre que cet algorithme est correct. (Le lecteur intéressé et consciencieux est invité à écrire les invariants associés aux trois boucles.)

10. Démonstration du Théorème. — Ce résultat n'est pas très difficle à obtenir; il faut seulement prendre soin de ne pas se perdre dans l'arborescence des démonstrations. Commençons par quelques remarques :

Par hypothèse, on a $m = p$ ou $\mathrm{ppd}(m) > p$, d'où $m \geq p$.

L'ensemble T contient tous les nombres premiers $p \leq N$ puisque on ne supprime que des entiers composés dans l'intervalle $[2\,..\,N]$.

Les entiers composés w qui sont supprimés satisfont $w \leq n$, donc aussi $\mathrm{ppd}(w) \leq n$. Montrons que $m \in T$. Si $m = p$ est premier, c'est terminé; si $m > p$, on a $\mathrm{ppd}(m) > p$ ce qui prouve $m > n$.

```
T := [2..N] ; p := 2 ;
while p^2 ≤ N do begin
    q := p ;
    while pq ≤ N do begin
        x := pq ;
        while x ≤ N do begin
            barrer(x, T) ;
            x := px
        end ;
        q := succ(q, T)
    end ;
    p := succ(p, T)
end
```

Crible d'Érathostène (version très intelligente).

Plus généralement, pour tout entier composé $((\pi, \mu, \lambda))$ appartenant à T, on a $p \leq \pi$, $\pi \in T$ et $\mu \in T$. Pour π, c'est évident. Si μ est premier, on a $\mu \in T$; si μ n'est pas premier, $\mathrm{ppd}(\mu) > \pi$ montre que $n \lessdot \mu$.

Passons à la démonstration proprement dite.

(i) Les inégalités $((p, m, \ell)) \lessdot ((p, m, \ell + 1)) = pn \leq N$ montrent que n a un successeur dans S_N. De $((p, m, \ell)) \lessdot ((\pi, \mu, \lambda)) \lessdot ((p, m, \ell + 1))$, on déduit $\pi = p$, puis $\mu = m$ et $\ell < \lambda < \ell + 1$. Donc pn est bien le successeur de n dans S_N.

(ii) Montrons que $T \cap]m, N]$ n'est pas vide. Le postulat de Bertrand affirme que l'on peut trouver un nombre premier q dans l'intervalle $]m, 2m[$. De $q < 2m \leq p^\ell m = n \leq N$, on déduit que $T \cap]m, N]$ contient q.

Terminons cette partie par deux remarques :

L'inégalité $((p, m, \ell)) \lessdot ((p, m, \lambda))$ exige $\ell < \lambda$. Par conséquent, on a $N < pn = p^{\ell+1} m \leq p^\lambda m$.

Le triplet $(p, m', 1)$ vérifie la condition (C), ce qui nous permet de parler du triplet $((p, m', 1))$. Si $\mathrm{ppd}(m') > p$, c'est terminé. Si $\mathrm{ppd}(m') = p$, il faut montrer que $m' = p$. Si ce n'est pas le cas, m' est composé et l'on peut écrire $m' = ((\pi, \mu, \lambda))$ avec $((p, m, \ell)) \lessdot ((\pi, \mu, \lambda))$ puisque $m' \in T$. Cette inégalité, jointe à l'hypothèse $\mathrm{ppd}(m') = \pi = p$, exige $m < \mu$ ou $m = \mu$ et $\ell < \lambda$. Or $\mu \in T$ et $\mu \leq \pi^\lambda \mu = m'$ et $m' = \mathrm{succ}(m, T)$ montrent que seule la condition $m = \mu$ et $\ell < \lambda$ est possible, d'où $m' = ((p, m, \lambda))$. Nous avons alors une contradiction d'après la première remarque de la partie précédente.

(iii) La majoration $n \lessdot ((p, m', 1))$ montre que n a un successeur dans S_N. Supposons que $((p, m', 1))$ ne soit pas ce successeur. La double inégalité

$((p,m,\ell)) \lessdot ((\pi,\mu,\lambda)) \lessdot ((p,m',1))$ exige $\pi = p$ et $m \leq \mu < m'$. Puisque $\mu \in T$, la minimalité de m' exige $m = \mu$ puis $\ell < \lambda$ ce qui est impossible, d'après (ii).

(iv) Si T contient encore un multiple ν de p, on a $\mathrm{ppd}(\nu) = p$ car $\mathrm{ppd}(\nu) < p$ implique ν barré. Par conséquent $\nu = ((p,\mu,\lambda))$ et $((p,m,\ell)) \lessdot ((p,\mu,\lambda))$. Si $m = \mu$ et $\ell < \lambda$, on a $N < pn = p^{\ell+1}m \leq p^{\lambda}\mu = \nu$, ce qui est impossible. Si $m < \mu$, et sachant que $\mu \in T$, la minimalité de m' exige $\mu \geq m'$, d'où l'on tire $p^{\lambda}\mu \geq p\mu \geq pm' > N$: nouvelle contradiction.

De $p \leq m < m'$ et $m' \in T$ on déduit que $p' = \mathrm{succ}(p,T)$ existe. Supposons alors p' composé. Comme T ne contient plus de multiples de p, nous pouvons d'écrire $p' = ((\pi,\mu,\lambda))$ avec $\pi > p$ et $\pi \in T$. Les inégalités $p < \pi < \pi^{\lambda}\mu = p'$ contredisent alors la minimalité de p'.

(v) L'inégalité $n = ((p,m,\ell)) \lessdot ((p',p',1)) = p'^2$ prouve que n possède un successeur dans S_N. Si p'^2 n'est pas ce successeur, on aurait $((p,m,\ell)) \lessdot ((\pi,\mu,\lambda)) \lessdot ((p',p',1))$. Cela exige $p = \pi$ ou $\pi = p'$. Le premier cas ne peut pas se produire, car T ne contient pas de multiples de p. Quant au deuxième cas, il exige $\pi = p' \leq \mu < p'$.

(vi) Supposons que n possède un successeur dans S_N. On aurait $((p,m,\ell)) \lessdot ((\pi,\mu,\lambda))$. Le cas $p = \pi$, $m = \mu$ et $\ell < \lambda$ ne peut pas se présenter car il exige $p^{\lambda}m > N$ d'après une remarque de (ii). Le cas $p = \pi$ et $m < \mu$ exige $m' \leq \mu$ (minimalité de m') ; la minoration $\pi^{\lambda}\mu \geq pm' > N$ montre que ce n'est pas possible. Reste le cas $\pi > p$. La minimalité de p' exige $\pi \geq p'$. De $\mu \geq \pi$, on déduit alors $\pi^{\lambda}\mu \geq p'^2 > N$, ce qui est absurde. □

11. Conclusion

Les performances sont-elles au rendez-vous? Absolument pas! (voir figure) ; le nouvel algorithme est environ *dix fois plus lent* que l'algorithme classique!

N	crible classique	crible très intelligent
5 000	1	13
10 000	2	26
20 000	4	52
30 000	6	79

Temps obtenus avec un Macintosh ; unité de temps = 1/60 de seconde.

La déception est grande... et l'explication facile à trouver : l'algorithme classique n'utilise que des additions, alors que l'algorithme intelligent n'utilise que des

multiplications, ce qui le pénalise très lourdement (un petit test confirme qu'une multiplication est de 10 à 20 fois plus lente qu'une addition selon la taille des entiers).

Moralité : en informatique aussi, le mieux est souvent l'ennemi du bien...

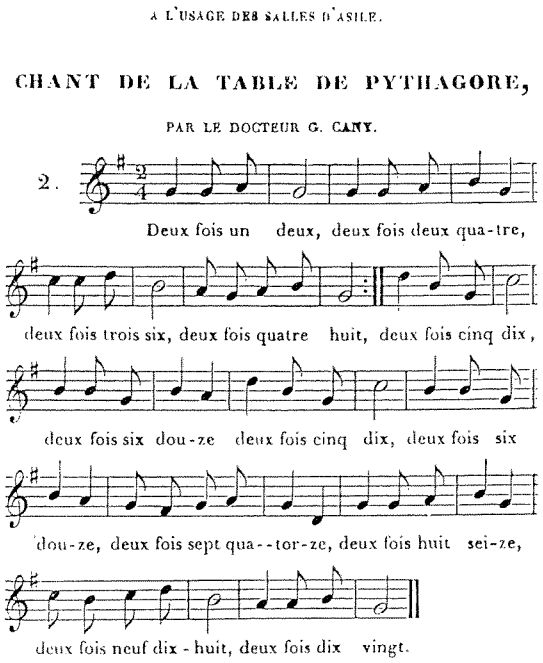
Post-scriptum. — Cette conclusion est volontairement provocatrice. La version très intelligente sert réellement à quelque chose. :

Tout d'abord, elle est très esthétique, ce qui suffit déjà à la justifier.

Mais il y a mieux : en utilisant des structures de données intelligentes qui permettent d'obtenir en un temps constant le premier élément non barré du tableau T, on peut démontrer que l'algorithme classique est un algorithme en $O(n \log n)$ alors que le deuxième algorithme est en $O(n)$. (Les temps expérimentaux sont cohérents avec ces résultats.) Mais ces résultats théoriques ne tiennent pas compte des vitesses respectives de l'addition et de la multiplication. (Bien entendu, il existe aussi des résultats théoriques faisant intervenir les complexités respectives de l'addition et de la multiplication.)

Références

GRIES (D.) and MISRA (J.). — *A Linear Sieve Algorithm for Finding Prime Numbers*, Comm. A.C.M., t. 21, 1978, p. 999–1003.



A L'USAGE DES SALLES D'ASILE.

CHANT DE LA TABLE DE PYTHAGORE,

PAR LE DOCTEUR G. CANY.

2.

Deux fois un deux, deux fois deux qua-tre,

deux fois trois six, deux fois quatre huit, deux fois cinq dix,

deux fois six dou-ze deux fois cinq dix, deux fois six

dou-ze, deux fois sept qua--tor-ze, deux fois huit sei-ze,

deux fois neuf dix - huit, deux fois dix vingt.