プログラミングの背景：数論

エラトステネスの篩

tbasic.org [*1]

[2020 年 6 月版]

> これらの数を作り出す方法はエラトステネスによって，篩と言われた。
> ・・・
> この篩法は以下の通りである。3 から始めて奇数を可能な限り順に作る。そして最初の項の倍数を順次取り除いていく。
> ・・・
> （ニコマコス「算術入門」）

素数は整数論の中でも最も基本的な対象です。素数の分布を調べるのは難しい問題ですが，比較的小さい数の素数をもとめたり，その表を作ることは，難しいことではありません。特にエラトステネスの篩を使えば素数表を効率的に作ることができます。

エラトステネスの篩は古代の方法ですが，その改良は，色々と行われてきました。特に計算機が普及し，計算効率への興味が高まる 20 世紀後半には，多くの研究成果が上がりました。ここでは，比較的近年に得られた結果も含めて，エラトステネスの篩，及びその改良篩による素数表の作成方法を説明します。アルゴリズム，例プログラムは tbasic で示しますが，比較的基本的な手順ですので，他の言語でも容易に書くことができるでしょう[*2]。

目次

[*1] http://www.tbasic.org

[*2] 詳しい出典等について興味のある人は，例えば，"Jonathan Sorenson, An introduction to prime number sieves, Computer Sciences Technical Report 909, Department of Computer Sciences, University of Wisconsin-Madison, January 1990" を参照してください。この文献は，2020 年現在インターネットで公開されています。

1　エラトステネスの篩の原理

エラトステネス（–275～–194）は 地球の大きさを測ったことで知られる古代ギリシアの学者です。この篩については，ニコマコス（１００頃）の「算術入門」第１巻１３章に篩法についての記述があり，エラトステネスが篩と言ったとあります。

それによると，この方法は，奇数の表

$$3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, ...$$

を作り，その中から，3 の倍数を順次除き，次に 5 の倍数を順次除き，一般に素数 p に対し，$f \cdot p$ の形の数 $(f > 1)$ を順次除きます。これらの操作を順次行い，残ったものが素数であるとするものです。

簡単な方法ですが，プログラムの問題としては基本的でよく扱われるものです。

1.1　素数とその性質

ここでは，素数とその基本的な性質から説明しましょう。

定義 1.1 (素数と合成数)．1 より大きい自然数 $2, 3, 4, \ldots$ の中で，真の約数を持たない数を**素数**（prime number）と言います。ここで真の約数とは，1 でもなく，その数自身でもない約数です。

これに対して，真の約数を持つ整数を**合成数**と言います。

2 や 3 は 1 とそれ自身以外の約数を持ちませんから，これらは素数です。他方，4 は 2 という真の約数を持ちますから，素数ではありません。すなわち合成数です。1 は特別な数で，素数でも合成数でもありません。

一般に，m が合成数とは，$1 < s < m, 1 < t < m$ となる自然数 s, t に対して

$$m = s \times t$$

と表されることを意味しています。逆に，素数はこのような分解が出来ないものとも言えます。

素数や合成数と言い方は，数が素数と言う元素によって乗法的に作られていると言う考え方の反映です。実際，2 以上の自然数は素数の積として表すことが出来ますが，その分解の仕方は順序の違いを無視すればただ一通りです。（この事実を素因数分解の一意性と言います。）

例えば，30 は

$$30 = 2 \times 3 \times 5$$

と表されますが，30 の素数の積への分解はこれ以外ありません。（勿論，３×2×5 や 5×2×3 と言った分解もありますが，これらは乗法の順序を変えただけなので，同じ分解と見ます。）ここで，30 は $2, 3, 5$ と言う数の元素（＝素数）を使って構成されていると見られる訳です。

次の性質は素数と合成数について基本的です。

命題 1.1. 自然数 n が合成数なら，$\sqrt{n}$ 以下の素因数を持つ[*3]。

[*3] 言い換えると，「合成数 n は $\sqrt{n}$ 以下の素数で篩われる」となります。

証明. n が合成数なら，$m \geqq k > 1$ となる整数に対して，$n = m \cdot k$ と表される。ここで，k の素因数を q とすると，q は n の素因数で，$q \leqq k \leqq m$ である。従って，

$$q^2 = q \cdot q \leqq q \cdot m \leqq k \cdot m = km = n$$

より，$q \leqq \sqrt{n}$ となる。 □

1.2 エラトステネスの篩いの原理

エラトステネスの篩の原理は素数を見つける代わりに合成数を除いていくものです。つまり，

エラトステネスの篩いの原理

1 より大きい数の表から合成数を除き，残ったものが素数

と言うものです。

大変簡単な原理ですから，100 以下の素数をすべて見つける位の問題なら，手で行っても簡単に実行できます。

また，多くの素数をこの方法を使って手計算で求めるのは大変ですが，コンピュータを使えば，かなり大きな数以下の素数をすぐに計算できます。

エラトステネスの篩は古代の方法ですが，現在でも素数表を作るときにはよく使われている方法です。

しかし，いくらコンピュータが高速でも大きな素数表を作る場合，効率は重要です。そこでここでは上の原理で素数表を作る場合の，なるべく効率的な具体的方法を考えましょう。考えるべき点は次です。

(1) どの数の倍数を除くか
(2) 数のどの倍数を取り除くか

以下これについて考えてみましょう。

(1) どの数の倍数を取り除くか

ある数の真の倍数はすべて合成数ですから，ある数 m に対してその 2 倍，3 倍,... と取り除いていきます，ここでもし m が合成数だとすると，m と m の倍数は既に取り除かれています。ですから，合成数の倍数は取り除く必要はありません。ですから，

素数の倍数のみ取り除けば良い

となります。真の倍数を取り除くことを篩うと言うことにします。この用語を使うと，このことは

素数でのみ篩えば良い

となります。

ところで，この篩う素数をどのようにして見つければ良いでしょうか。実はこれは極めて簡単です。今のように小さい方の素数から順に篩って行った場合，p まで篩ったとすると，p より大きい数で取り除かれていない最小の数は素数になります。実際，q が p より大きい数で取り除かれていない最小の数

とするとき，もし q が合成数なら，

$$q = p' \times r$$

となる q 未満の素数 p' が存在します。p と q の間には素数はありませんから，この p' は p 以下の素数です。p 以下の素数では既に篩われていますから，これは矛盾です。従って q は素数です。ですから，

> 次に篩う素数は，今まで篩った素数より大きい，表に残っている最初の数

になります。

(2) 数のどの数の倍数を取り除くか

上の考察から，素数 p で篩うことを考えます。

素数 p で篩う場合，$2p, 3p, \ldots$ と取り除いていくわけですが，$m < p$ の場合，mp は m の倍数ですから，既に取り除かれています。ですから，実際に取り除く必要があるのは，$p \cdot p$，$(p+1) \cdot p, \ldots$ となります。即ち，

> 素数 p に対して，
>
> $$p \cdot p, \quad (p+1) \cdot p, \quad (p+2) \cdot p, \quad (p+3) \cdot p, \quad (p+4) \cdot p, \quad \ldots \qquad \text{(基本篩法)}$$
>
> と p 以上の倍数を取り除けばよい

ことになります。

更に，もし p が 3 以上なら，$p+1$ は偶数で，$(p+1) \cdot p$ は既に取り除かれています。ですから，

> p が 3 以上のときは，
>
> $$p \cdot p, \quad (p+2) \cdot p, \quad (p+4) \cdot p, \quad (p+6) \cdot p, \quad (p+8) \cdot p, \quad \ldots \qquad \text{(改良篩法)}$$
>
> と p 以上の奇数倍を取り除けばよい

ことになります。

また，n 以下の素数表を作る場合，n の平方根以上の素数 q を使って篩うものは，$q^2 > n$ となりますから，篩うべき数は表にありません。ですから，

> n 以下の素数表を作る場合，$p < \sqrt{n}$ となる素数 p についてのみ篩えば良い

ことが分かります。

この最後の事実は命題 1.1 と実質的に同じ内容です。

ここで以後の説明を明確にするために，用語を確認しましょう。

定義 1.2 (篩う数，篩われる数)**.** 素数 p で篩うとき，p を**篩う数**と言う。p により取り除かれる p より大きい p の倍数[*4]のことを p で**篩われる数**と言う。mp が篩われる数のとき，m を p で篩うときの乗数という。

[*4] 数学で p の倍数と言うと，普通 p も含まれます。

2　エラトステネスの篩での計算

2.1　100 以下の素数の計算

エラトステネスの篩を使って，100 以下の素数を全て求めてみましょう。

まず，2 以上 100 以下の数の表を作ります。

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

(表 0)

そこで 2 で篩います。即ち，2 より大きい 2 の倍数を除きます。除いた数に取り消し線 ~~ を書きます。$50 \times 2 = 100$ が最後です。

	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~
21	~~22~~	23	~~24~~	25	~~26~~	27	~~28~~	29	~~30~~
31	~~32~~	33	~~34~~	35	~~36~~	37	~~38~~	39	~~40~~
41	~~42~~	43	~~44~~	45	~~46~~	47	~~48~~	49	~~50~~
51	~~52~~	53	~~54~~	55	~~56~~	57	~~58~~	59	~~60~~
61	~~62~~	63	~~64~~	65	~~66~~	67	~~68~~	69	~~70~~
71	~~72~~	73	~~74~~	75	~~76~~	77	~~78~~	79	~~80~~
81	~~82~~	83	~~84~~	85	~~86~~	87	~~88~~	89	~~90~~
91	~~92~~	93	~~94~~	95	~~96~~	97	~~98~~	99	~~100~~

次に 2 の次で最初に残っている 3 で篩います。この場合，(基本篩法) では 3 の倍数

$$3 \cdot 3 = 9,\ 4 \cdot 3 = 12,\ 15,\ 18,\ 24,\ 27,\ 30,\ 33,\ 36,\ 39,\ 42,\ 45,\ 48,$$
$$51,\ 54,\ 57,\ 60,\ 63,\ 66,\ 69,\ 72,\ 75,\ 78,\ 81,\ 84,\ 87,\ 90,\ 93,\ 96,\ 99$$

を取り除きます。$33 \times 3 = 99$, $34 \times 3 = 102$ ですから，33×3 が最後です。しかし，よく見ると，3 の偶数倍

$$4 \cdot 3 = 12,\ 6 \cdot 3 = 18,\ 8 \cdot 3 = 24,\ 30,\ 36,\ 42,\ 48,\ 54,\ 60,\ 66,\ 72,\ 78,\ 84,\ 90,\ 96$$

は既に取り除かれていることが分かります。ですから，(改良篩法) で，

$$3 \cdot 3 = 9,\ 5 \cdot 3 = 15,\ 7 \cdot 3 = 21,\ 9 \cdot 3 = 27,\ 11 \cdot 3 = 33,\ 39,\ 45,\ 51,\ 57,\ 63,\ 69,\ 75,\ 81,\ 87,\ 93,\ 99$$

と 3 の奇数倍を取り除いても結果は同じです。

除いた数に取り消し線 == を書くと，次が得られます。

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	35	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	49	~~50~~
~~51~~	~~52~~	53	~~54~~	55	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	65	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	77	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	85	~~86~~	~~87~~	~~88~~	89	~~90~~
91	~~92~~	~~93~~	~~94~~	95	~~96~~	97	~~98~~	~~99~~	~~100~~

次に 3 の次で最初に残っている 5 で篩います。この場合，(基本篩法) でも良いのですが，結果として (改良篩法) で

$$5 \cdot 5 = 25,\ 7 \cdot 5 = 35,\ 9 \cdot 5 = 45,\ 11 \cdot 5 = 55,\ 55,\ 65,\ 75,\ 85,\ 95$$

と 5 の奇数倍を取り除きます。$19 \times 5 = 95,\ 21 \times 5 = 105$ ですから，19×5 が最後です。除いた数に取り消し線 ═を書くと，次が得られます。

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	49	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	77	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
91	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

次に 5 の次で最初に残っている 7 で篩います。この場合も，(基本篩法) でも良いのですが，結果として (改良篩法) で

$$7 \cdot 7 = 49,\ 9 \cdot 7 = 63,\ 11 \cdot 7 = 77,\ 13 \cdot 7 = 91$$

と 7 の奇数倍を取り除きます。$13 \times 7 = 91,\ 15 \times 7 = 105$ ですから，13×7 が最後です。除いた数に取り消し線 ═ を書くと，次が得られます。

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

7 の次に残っている最初の数は 11 ですが，$11 \times 11 = 121 > 100$ ですから，これで篩は終了です。ここで

残っている 25 個

	2	3	5	7	
11		13		17	19
		23			29
31				37	
41		43		47	
		53			59
61				67	
71		73			79
		83			89
				97	

が 100 以下の素数になります。

2.2 プログラム例

以上のことを行うプログラムを書いてみましょう。

表と配列変数を対応させます。配列変数名として，NT [*5] を使用することにします。そこで，100 までの数表を使うとして，配列変数 Dim NT(100) と宣言します。[*6]

NT(1)〜NT(100) と前項 (表0) との対応は，NT(i) は (表0) での数 i と対応し，NT(i) の値は数 i についての情報を持つものとします。

具体的には，ある数で篩った結果，自然数 m が取り除かれる場合，NT(m) = 1 と設定し，m が合成数であることを示します[*7]。

[基本篩法] によるプログラム

p を素数とするとき，(基本篩法) では，

$$p \cdot p,\ (p+1) \cdot p,\ \ldots,\ m \cdot p,\ \ldots$$

と p 以上の p の倍数で篩います。m を p で篩う乗数とします。篩う表の数値の最大値は 100 ですから，m は $m \cdot p \leqq 100$ を満たします。この条件を満たす最大値を求めるために次の事実に注意します。

命題 2.1. M, k を正整数とする。$\dfrac{M}{k}$ の整数部分を $\left[\dfrac{M}{k}\right]$ と表すと[*8]，

$$\left[\frac{M}{k}\right] \cdot k \leqq M < \left(\left[\frac{M}{k}\right] + 1\right) \cdot k$$

となる。

[*5] **N**umber **T**able のつもりです。

[*6] 初期値として，NT(i) = 0 となっています。

[*7] 以後，DimNT(MaxN) で宣言された配列変数 NT は，このような意味を設定する 1〜MaxN の篩う数表を意味するものとします。上の場合は MaxN = 100 です。

[*8] 一般に実数 a に対し，その整数部分を $[a]$ と表します。この記法はガウス記号と言われています。この関数は tbasic では int 関数で実現されます。
また，M, k が正整数の場合は，Int(M/k) は M¥k とも書けます。

証明. $q = \left[\frac{M}{k}\right]$ とすると，$\frac{M}{k} = q + r\ (0 \leqq r < 1)$ と表される。従って，

$$\left[\frac{M}{k}\right] \cdot k = qk \leqq qk + rk = M < qk + k = (q+1)k = \left(\left[\frac{M}{k}\right] + 1\right) \times p$$

となる。 □

この命題を特に，$M = 100,\ k = p$ について適用すると，

$$\left[\frac{100}{p}\right] \cdot p \leqq 100 < \left(\left[\frac{100}{p}\right] + 1\right) \cdot p$$

が得られます。従って，乗数 m の最大値は $\left[\frac{100}{p}\right]$ で，m は，

$$m = p,\ p+1,\ p+2,\ \ldots,\ \left[\frac{100}{p}\right]$$

の範囲を取ります。この篩処理は

```
For i=p To Int(100/p)
   NT(i*p)=1
Next i
```

で実現されます。

素数 p の次に篩う素数 q は，

$$p+1,\ p+2,\ p+3,\ \ldots\ p+k,\ \ldots$$

の中で，最初に現れる NT(p + k) = 0，即ち，NT(p + k) ≠ 1 となるものですから，

```
p=p+1
While NT(p)=1:p=p+1: End While
```

で得られます[*9]。

以上を纏めて一つのプログラムとすると次のようになります。

プログラム **2.1**（基本篩）

```
'-----------------------------------
' 100 以下の素数表の作成
'-----------------------------------
Dim NT(100)
p=2
While p*p<=100
   For i=p To Int(100/p)
      NT(i*p)=1
   Next i
   p=p+1
   While NT(p)=1:p=p+1: End While
End While
For i=2 To 100
   If NT(i)=0 Then Print i
Next i
End
'-----------------------------------
```

[*9] ここで，While ～ End While 文を使っています。この構文は While ～ Wend 文と全く同じですが，Visual Basic ではこれを使いますので，ここでは互換性のために，While ～ End While 文を使いました。tbasic では両方使えます。

このプログラムは短く，内容も分かりやすい簡単なものです。

また，100 のところを書き換えることで，100 以上の素数表を作るプログラムにすることができます。次のプログラムは MaxN までの素数表を作成するものです。

プログラム **2.1'**（基本篩）

```
'-----------------------------------
' MaxN 以下の素数表の作成
'-----------------------------------
Const MaxN=1000
Dim NT(MaxN)
p=2
While p*p<=MaxN
   For i=p To Int(MaxN/p)
      NT(i*p)=1
   Next i
   p=p+1
   While NT(p)=1: p=p+1: End While
End While
For i=2 to MaxN
   If NT(i)=0 Then Print i
Next i
End
'-----------------------------------
```

上のプログラムでは Const MaxN = 1000 となっていますが[*10]，この部分を書き換えることで，100000000 位の素数表を作ることもできます[*11]。

[改良篩法**]** によるプログラム

プログラム2.1′ は簡単なプログラムですが，かなり実用的です。実際，1000000 程度の素数表であれば，数秒で計算可能ですし，100000000 までの素数表であっても数分で計算できます。ですから，実際問題として，素数表を作る必要がある場合は，プログラム 2.1' でかなりの場合に間に合うと思われます。

それでも，より効率の良い方法があればと思うかもしれません。改良篩法 は 基本篩法 を改良した方法で，基本篩法 よりも少し効率的です。

ここでは，改良篩法 を使ったプログラムを考えてみましょう。

まず，2 で篩いますが，この部分は 基本篩法 と同じです。$p = 2$ として

```
For i=2 To 100/2
   NT(i*2)=1
Next i
```

が得られます。

次に p を奇素数し，p で篩います。(改良篩法) では，

$$p \cdot p,\ (p+2) \cdot p,\ (p+4) \cdot p,\ (p+6) \cdot p,\ \ldots\ ,\ r \cdot p$$

[*10] ここで Const 宣言をする理由は，tbasic の仕様として，Dim 宣言で使える数は定数のみということによります。MaxN を普通変数として利用して，

```
MaxN = 1000
Dim NT(MaxN)
```

とするとエラーになります。

[*11] 大きな MaxN を使う場合，結果の表示に膨大な時間が掛かることもありますから，それについての対処が必要です。

と p 以上の p の奇数倍数で篩います。

ここで，命題 2.1 から

$$\left[\frac{100}{p}\right] \times p \leqq 100 < \left(\left[\frac{100}{p}\right] + 1\right) \times p$$

が得られます。従って，

$$r \leqq \left[\frac{100}{p}\right] \qquad (*)$$

となる最大の奇数 r が乗数の最大値になります。これは

```
For i=p To Int(100/p) Step 2
   NT(i*p)=1
Next i
```

と書くことができます[*12]。

奇素数 p の次に篩う奇素数 q を見つけるのは，

$$p+2,\ p+4,\ p+6,\ \ldots,\ p+2k,\ \ldots$$

の中で，最初に現れる N(p + 2k) ≠ 1 となるものですから，

```
p=p+2
While NT(p)=1:p=p+2: End While
```

で得られます。

以上を纏めて一つのプログラムとすると次のようになります。

プログラム **2.2**（改良篩）

```
'-----------------------------------
' 100 以下の素数表の作成
'-----------------------------------
Dim NT(100)
For i=2 To 100/2
   NT(i*2)=1
Next i
p=3
While p*p<=100
   For i=p To Int(100/p) Step 2
      NT(i*p)=1
   Next i
   p=p+2
   While NT(p)=1:p=p+2: End While
End While
For i=2 To 100
   If NT(i)=0 Then Print i
Next i
End
'-----------------------------------
```

[*12] For 文の部分は，r を $(*)$ を満たす奇数の最大数とするとき，For i=p To r Step 2 とするのが正確ですが，For 文のループは i ≦ r の範囲で実行されますから，ここでの記述で目的の動作をします。

上のプログラム 2.2 はプログラム 2.1 またはプログラム 2.1' に比べて $p = 2$ について場合分けをしていますから，少しだけ複雑になりますが，高速化が期待されます。実際，p が奇素数の場合は，(基本篩法) に比べて処理回数が半分になります。

プログラム 2.2 も 100 のところを書き換えることで，より大きな素数表を作ることのできる次のプログラム 2.2' になります。

プログラム **2.2'**（改良篩）

```
'-----------------------------------
' MaxN 以下の素数表の作成
'-----------------------------------
Const MaxN=1000
Dim NT(MaxN)
For i=2 To Int(MaxN/2)
   NT(i*2)=1
Next i
p=3
While p*p<=MaxN
   For i=p To Int(MaxN/p) Step 2
      NT(i*p)=1
   Next i
   p=p+2
   While NT(p)=1:p=p+2: End While
End While
For i=2 to MaxN
   If NT(i)=0 Then Print i
Next i
End
'-----------------------------------
```

私の環境[*13]で，プログラム 2.2 とプログラム 2.2 の実行速度を比較してみた結果は次の通りです。

MaxN	10^6	$5 \cdot 10^6$	10^7	$5 \cdot 10^7$	10^8
プログラム2.1'(基本篩)	1秒	5秒	12秒	59秒	2分
プログラム2.2'(改良篩)	1秒	4秒	9秒	44秒	1分32秒

この結果によれば，MaxN = 10^8 の場合，プログラム 2.2' はプログラム 2.1' の約 $\frac{3}{4}$ の時間で計算できたことになります。

プログラム 2.1' あるいはプログラム 2.2' は素数表作成のために十分有用なプログラムですが，それでも限界があります。大きな素数表を作成するには多少の時間が必要です。

そこで，次節では計算効率を改良して，より高速で素数表を作成をすることを考えてみましょう。

[*13] 私の現在の環境は，エントリーと思えるデスクトップマシンで，OS は 64 ビット Windows 10 です。以後いくつかのプログラムの実行時間について書きますが，すべて私の現在の環境でのものです。ハイエンドモデルならこの数倍の速度が出るでしょう。また，tbasic はインタプリタですので，そもそも実行速度は速くありません。コンパイラ言語でプログラムを作れば 10 倍以上の速度が出ると思います。ですから，絶対的な実行時間として見るというより，いくつかの方法の比較のための目安としてください。

3　車輪の利用

エラトステネスの篩は古くからある計算法ですが，効率化への多くの工夫が提案されています。前節で説明した改良篩法は最も古い効率化への工夫でした。

計算機が普及し，計算効率についての興味が高まる 20 世紀後半には，特に多くの研究がなされました。ここでは，以後それらの成果を説明します。

3.1　車輪

100 以下の素数表を求める場合を再考してみましょう。まず，2 以上 100 以下の数の表を作ります。これが，(表0) ですが，これを 2 で篩った結果が次の通りでした。ここでは，除いた数に取り消し線 ── を引いています。

	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~
21	~~22~~	23	~~24~~	25	~~26~~	27	~~28~~	29	~~30~~
31	~~32~~	33	~~34~~	35	~~36~~	37	~~38~~	39	~~40~~
41	~~42~~	43	~~44~~	45	~~46~~	47	~~48~~	49	~~50~~
51	~~52~~	53	~~54~~	55	~~56~~	57	~~58~~	59	~~60~~
61	~~62~~	63	~~64~~	65	~~66~~	67	~~68~~	69	~~70~~
71	~~72~~	73	~~74~~	75	~~76~~	77	~~78~~	79	~~80~~
81	~~82~~	83	~~84~~	85	~~86~~	87	~~88~~	89	~~90~~
91	~~92~~	93	~~94~~	95	~~96~~	97	~~98~~	99	~~100~~

(表 0-1)

この (表0 – 1) を $p = 3$ に対して，(基本篩法) で篩います。ここでは，除いた数に取り消し線 ──，既に取り消し線が引かれているところ（除かれている数）には二重取り消し線 ═ を書くことにします。すると，次が得られます。

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	35	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	49	~~50~~
~~51~~	~~52~~	53	~~54~~	55	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	65	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	77	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	85	~~86~~	~~87~~	~~88~~	89	~~90~~
91	~~92~~	~~93~~	~~94~~	95	~~96~~	97	~~98~~	~~99~~	~~100~~

(表 0-11)

3 の偶数倍

12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90, 96　　(*)

が既に取り除かれていることが分かります。これらは，今の場合は篩う必要がないものでした。このようにエラトステネスの篩では，既に篩われているものを，更に篩うことがよくあります。(改良篩法) は p が 3 以上のとき，p の乗数 m を奇数に限ることで，(∗) の繰り返しを防ぐ工夫でした。この (表0 – 1) を $p = 3$ に対して，今度は (改良篩法) で篩うと，次が得られます。

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	35	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	49	~~50~~
~~51~~	~~52~~	53	~~54~~	55	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	65	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	77	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	85	~~86~~	~~87~~	~~88~~	89	~~90~~
91	~~92~~	~~93~~	~~94~~	95	~~96~~	97	~~98~~	~~99~~	~~100~~

(表 0-21)

今度は二重取り消し線 ═ が無く，二重に篩うところはありませんでした。

しかし，この (表0 – 21) を $p = 5$ に対して，(改良篩法) で篩うと，

$$5 \cdot 5 = \underline{25},\ 7 \cdot 5 = \underline{35},\ 9 \cdot 5 = \underline{45},\ 11 \cdot 5 = \underline{55},\ 13 \cdot 5 = \underline{65},\ 15 \cdot 5 = \underline{75},\ 17 \cdot 5 = \underline{85},\ 19 \cdot 5 = \underline{95}$$

が篩われ，次が得られます。

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	49	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	77	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
91	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

5 の倍数で

$$45 = 3 \cdot 3 \cdot 5,\ 75 = 3 \cdot 5 \cdot 5$$

に二重取り消し線 ═ が引かれています。これらの数はいずれも乗数として，$p = 5$ より小さい素数 3 を因数に持っています。

一般に，素数 p で篩う場合，乗数が p より小さい素数を含むと，既に篩われているので，重複して篩われることになります。この重複を防ぐ方法として，

(1) 重複する乗数を判定して，重複しないものについて篩う
(2) 重複しない乗数を求めて，それで篩う。

が思いつきます。(1) の方法は例えば，M を p 未満の素数の積とするとき，

```
For i=p To Int(MaxN/p)
   If GCD(M,i)=1 Then NT(i*p)=1
Next i
```

のようなことが考えられます。しかし，これは余り効率的ではありません。実際，これは何もしない

```
For i=p To Int(MaxN/p)
   NT(i*p)=1
Next i
```

とほぼ同じ計算処理が必要になります。ですから，(1) のアイデアでは目的を達成できません。

(2) の方法は (改良篩法) で $p \geqq 3$ に対して，

```
For i=p to Int(MaxN/p)  step 2
   NT(i*p)=1
Next i
```

とある部分に例があります。p が奇素数なので，

```
For i=p To Int(MaxN/p)  step 2
```

で i は奇数を動きます。即ち，2 と互いに素な数を動き，i の奇数判定は行っていません。ですからこれは効率的な方法と言えます。

これと類似なことを，$2 \cdot 3 = 6$ と互いな素な数で行うことを考えてみましょう。

例 **3.1.** 6 と互いに素な数は 6 を法にして決定されます。そして，6 を法にして，6 と互いに素なものは，1 (mod 6) と 5 (mod 6) です。従って，$p = 5$ から始めたとすると，6 と互いに素なものは，

$$\underline{5} = 0 \cdot 6 + 5,\ \underline{7} = 1 \cdot 6 + 1,\ \underline{11} = 1 \cdot 6 + 5,\ \underline{13} = 2 \cdot 6 + 1,\ \underline{17} = 2 \cdot 6 + 5, \ldots$$

と続きます。この数列は，その差に注目すると

$$\underline{5} = 1 + 4,\ \underline{7} = 5 + 2,\ \underline{11} = 7 + 4,\ \underline{13} = 11 + 2,\ \underline{17} = 13 + 4, \ldots$$

と書けます。即ち，順次 4，2 を加えることで次の数を得ることができます。

従って，この数列は，配列変数 $\mathtt{w_2(0)}, \ldots \mathtt{w_2(5)}$ を使い[*14]，

i	0	1	2	3	4	5
$\mathtt{w_2}(i)$	0	4	0	0	0	2

と設定して，$\mathtt{i} = \mathtt{i} + \mathtt{w_2}(\mathtt{i} \bmod 6)$ と表すことができます。このことを使うと，6 と素な乗数による篩を次のように表すことができます。

```
i=p
While i<= Int(MaxN/p)
   NT(i*p)=1
   i =  i + w2(i mod 6)
End While
```

この処理では，乗数が 6 と素であるかの判定は行わず，6 と素な乗数を直接求めて，計算処理をしています。即ち (2) の方法になっています。

このことを一般化して，車輪配列 $\mathtt{w_k}$ を次のように定義します。

[*14] 配列変数をプログラムの中で使う場合，$\mathtt{w_2(0)}$ での $_2$ のような添え字は使えませんが，この場合 2 が動きうる数であることを示すために，添え字の形としました。実際のプログラムでは w2(0) のような形になります。以下でもこのような添え字の使い方することがあります。

定義 3.1 (車輪配列 $\mathtt{w_k}$)**.** k を正整数とし，p_k を k 番目の素数とする。$p_1 = 2$ から p_k をまでの積を $\mathtt{mP_k}$ と表す。即ち，$\mathtt{mP_k} = 2 \cdot 3 \cdot 5 \cdots p_k$ とする。$\mathtt{w_k}(0)$〜$\mathtt{w_k}(\mathtt{mP_k} - 1)$ を整数の配列変数とし，$0 \leqq i < \mathtt{mP_k}$ に対して，$\mathtt{w_k}(i)$ は次の性質を持つとする。

(1) $\gcd(i, \mathtt{mP_k}) > 1$ ならば，$\mathtt{w_k}(i) = 0$
(2) $\gcd(i, \mathtt{mP_k}) = 1$ ならば，$\mathtt{w_k}(i)$ は $\mathtt{mP_k}$ と互いに素な i の次の数との差を表す。
　従ってこのとき，$i + \mathtt{w_k}(i)$ は $\mathtt{mP_k}$ と互いに素な i の次の数を表す。

このとき，配列 $\mathtt{w_k}(i)$ を k 番目の**車輪 (wheel) 配列**という[*15]。ここで，$\mathtt{w_0}$ は定義されていませんが，$\mathtt{mP_0} = 1$ として，$\mathtt{w_0}(0) = 1$ と定めることとします。

この車輪配列について，定義から，次が成立します。

命題 3.1. i を $\mathtt{mP_k}$ と互いに素な正整数とすると，

$$i + \mathtt{w_k}(i \bmod \mathtt{mP_k})$$

は $\mathtt{mP_k}$ と互いに素な i の次の正整数を表す[*16]。

証明. i を $\mathtt{mP_k}$ で割って，商を q，余りを i_0 とする。即ち，$i = q \cdot \mathtt{mP_k} + i_0$ $(0 \leqq i_0 < \mathtt{mP_k})$ とする[*17]。このとき，

$$i + \mathtt{w_k}(i \bmod \mathtt{mP_k}) = q \cdot \mathtt{mP_k} + i_0 + \mathtt{w_k}(i_0)$$

となる。ここで，$i_0 + \mathtt{w_k}(i_0)$ は $\mathtt{w_k}$ の定義から，$\mathtt{mP_k}$ と互いに素な i_0 の次の正整数である。従って，$q \cdot \mathtt{mPk} + i_0 + \mathtt{w_k}(i_0) = i + \mathtt{w_k}(i_0)$ は $\mathtt{mP_k}$ と互いに素な正整数である。
　更に，$0 < r < \mathtt{w_k}(i_0)$ となる整数 r に対して，$i + r = q \cdot \mathtt{mP_k} + i_0 + r$ が $\mathtt{mP_k}$ と互いに素であったとすると，$i_0 + r$ も $\mathtt{mP_k}$ と互いに素であり，$i_0 + \mathtt{w_k}(i_0)$ が i_0 の次の $\mathtt{mP_k}$ と互いに素であるという $\mathtt{w_k}(i_0)$ の定義に反する。　□

この車輪配列を利用すれば，p を p_{k+1} 以上の素数とするとき，一般に

```
'------------------------------
' 車輪配列 wk による篩の擬似コード
'------------------------------

i = p
While i <= Int(MaxN/p)
      NT(i * p) = 1
      i = i + wk(i mod mPk)
End While
'------------------------------
```

によって $\mathtt{mP_k}$ と互いに素な数を篩うことができます。(基本篩法) は，$\mathtt{w_0}$ による篩と考えることもできます。

[*15] 車輪という用語は，篩う数を $\mathtt{mP_k}$ を法として巡回的に決定することに由来します。
[*16] $0 < i < \mathtt{mP_k}$ の場合は，この性質は，$\mathtt{w_k}$ の定義でした。
[*17] ここで，仮定から，i は $\mathtt{mP_k}$ と互いに素ですから，実際は $0 < i_0$ です。

車輪を使った篩の効率について，次が成立します。

命題 3.2. $\mathtt{w}_0$，$\mathtt{w}_1$，$\mathtt{w}_2$，...，$\mathtt{w}_k$，... に対して順次 p_1，p_2，p_3，...，p_{k+1} で篩っていくとする。このとき，

(1) 車輪 $\mathtt{w}_k$ を使って p_{k+1} で篩ったとき，篩の重複は起こらない。
(2) 車輪 $\mathtt{w}_k$ を使って p_{k+2} で篩ったとき，篩の重複が起こる。

証明. (1) 車輪 $\mathtt{w}_k$ を使って p_{k+1} で篩うと，乗数 f に対して，篩う数は $f \cdot p_{k+1}$ の形である。ここで，乗数は $i + \mathtt{w}_k(i \bmod \mathtt{mP}_k)$ の形だから命題 3.1 より，f は $\mathtt{mP}_k$ と互いに素になる。更に，p_{k+1} も $\mathtt{mP}_k$ と互いに素だから，$f \cdot p_{k+1}$ も $\mathtt{mP}_k$ と互いに素になる。一方 $\mathtt{w}_0$，$\mathtt{w}_1$，...，$\mathtt{w}_{k-1}$ は，それぞれ，p_1，p_2，...，p_k で篩うものだから，それらで篩った数は，$\mathtt{mP}_k = p_1 p_2 \cdots p_k$ と互いに素ではない。従って，$f \cdot p_{k+1}$ と同じものはない。

(2) $p_{k+1}^n > p_{k+2}$ となる自然数 n を取る[*18]。このとき，

$$p_{k+1}^{n-1} \cdot p_{k+2} \cdot p_{k+1} = p_{k+1}^n \cdot p_{k+2} \qquad (*)$$

である。ここで，$p_{k+1} \leqq p_{k+1}^{n-1} \cdot p_{k+2}$ で，$p_{k+1}^{n-1} \cdot p_{k+2}$ は $\mathtt{mP}_k$ と互いに素だから，$p_{k+1}^{n-1} \cdot p_{k+2}$ は $\mathtt{w}_k$ を使った p_{k+1} による篩の乗数になる。他方，$p_{k+2} \leqq p_{k+1}^n$ だから，同様に，p_{k+1}^n は $\mathtt{w}_k$ を使った p_{k+2} による篩の乗数になる。従って，$(*)$ の左辺は p_{k+1} で篩われた数を表し，右辺は p_{k+2} で篩われた数を表す。即ち，$(*)$ は重複され篩われる数になる。

□

$k = 0, 1, 2, 3$ について具体的に，$\mathtt{w}_k$ を計算すると次になります。

例 3.2. (1) $k = 0$ のとき，$\mathtt{mP}_0 = 1$ で $\mathtt{w}_0$ は次のように定める。

i	0
$\mathtt{w}_0(i)$	1

(2) $k = 1$ のとき，$\mathtt{mP}_1 = 2$ で $\mathtt{w}_1$ は定義から次になる。

i	0	1
$\mathtt{w}_1(i)$	0	2

(3) $k = 2$ のとき，$\mathtt{mP}_2 = 2 \cdot 3 = 6$ で $\mathtt{w}_2$ は定義から次になる。

i	0	1	2	3	4	5
$\mathtt{w}_2(i)$	0	4	0	0	0	2

(4) $k = 3$ のとき，$\mathtt{mP}_3 = 2 \cdot 3 \cdot 5 = 30$ で $\mathtt{w}_3$ は次になる。

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\mathtt{w}_3(i)$	0	6	0	0	0	0	0	4	0	0	0	2	0	4	0
i	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
$\mathtt{w}_3(i)$	0	0	2	0	4	0	0	0	6	0	0	0	0	0	2

このように $\mathtt{w}_3$ の場合，$\mathtt{mP}_k = 30$ と互いに素な数（0〜29) は 8 個で，それらの比は $8/30 = 0.26\ldots$ となります。即ち，30 と互いに素な数は全体の約 1/4 です。車輪を使うと，これら 30 と互いに素な数のみで篩うことになります。つまり車輪を利用することで，約 3/4 の計算を節約することができます。

[*18] 実は，$n = 2$ で十分です。（ベルトラン＝チェビシェフの定理）

$w_k(i)$ のデータを求めるには，小さい mP_k に対しては手計算でも十分可能ですが，mP_k は急激に大きくなります。例えば，$k = 5$ の場合は $mP_5 = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 = 2310$ となり，手計算では煩雑になります。

しかし，一般に $w_k(i)$ のデータは定義から，例えば，次の作成プログラム使って簡単に求めることができます。

mP_k と互いに素な最初の数は 1 ですが，その次，$i = 2$ から始めて，2 番目の mP_k と互いに素な次の数が見つかったとき，その数と一つ前の mP_k と互いに素な数との差を，一つ前の数の位置に格納します。この処理を順次 $i = mP_k - 1$ まで続けます。$w_k(mP_k - 1)$ は常に 2 なので[*19]，最後に設定します。

以上をプログラムとすると次が得られます。pt は mP_k と互いに素な数を表します。これを使えば大きな k に対しても w_k が簡単に作成できます。

```
'------------------------------
' 車輪配列 wk 作成擬似コード
'------------------------------

pt = 1
For i = 2 to mPk - 1
      If GCD(i,mPk) = 1 then
            wk(pt) = i - pt
            pt = i
      End If
Next i
wk(mPk - 1) = 2

'------------------------------
```

[*19] $(mP_k - 1) + 2 = mP_k + 1$ は mP_k と互いに素です。

3.2 車輪を利用した篩

前節では車輪について説明しました。ここでは，車輪を利用して篩のプログラムを作成してみましょう。

プログラム 2.2 は，w_1 を使ったプログラムと見ることができます。そこで，w_2 をそれに追加しましょう。

w_2 は $p_3 = 5$ 以上の素数に対しての篩法ですから，プログラム 2.2 に，$p_3 = 5$ 以上の部分に，上述の「車輪配列 wk による篩の擬似コード」を追加すれば出来上がりです。

```
'-----------------------------------
' 100 以下の素数表の作成
'-----------------------------------
Dim NT(100)
Dim w2(5)
w2(1)=4: w2(5)=2

For i=2 To 100/2
   NT(i*2)=1
Next i

' 実質 w1 を使った篩
p=3
For i=p To Int(100/p) step 2
   NT(i*p)=1
Next i

' w2 を使った篩
p=5
While p*p<=100
   i=5
   While i<= Int(100/p)
      NT(i*p)=1
      i =  i + w2(i mod 6)
   End While
   p=p+2
   While NT(p)=1:p=p+2:End While
End While
For i=2 To 100
   If NT(i)=0 Then Print i
Next i
End
'-----------------------------------
```

車輪を使った篩は，大きな車輪を使うほど効率が上がります。しかし上のプログラムの方法だと，w_k の車輪を使うには，w_0 から始めて，順次 w_1，w_2，...，w_{k-1} と篩う必要があります。w_k での篩は，$p_1, \ldots p_k$ で篩った残りの数を篩うものですから，w_k での篩を行うには，$p_1, \ldots p_k$ で篩われた数を NT(i) から取り除く必要があるからです。

これは，

> mP_k と素でない数 i に対し，NT(i) = 1 とし，$i = 1, \ldots k$ に対して，NT(p_k) = 0 とする

ことで実現できます。この設定によって w_0，w_1，...，w_{k-1} の篩に代えることができます。

他方，車輪配列 w_k の性質から，mP_k と互いに素でない i に対しては，w_k(i) = 0 となっています。このことを利用すると，比較的簡単に上のことが実現できます。

mP_k と素でない数 i に対し，NT(i) = 1 とする，最も単純な方法として，

```
For i=1 To MaxN
   If wk(i mod mPk)=0 Then NT(i)=1
Next i
```

が考えられます。しかし，これは MaxN 個の i について判定処理をしているため，余り効率が良くありません。ここで，車輪配列 w_k が mP_k を法にして定まることに注意すれば

```
'------------------------
' w0～wk-1篩の代替処理
'------------------------
For i=0 To mPk-1
   If wk(i)=0 then
      j=i
      While j<=MaxN
         NT(j)=1
         j=j+mPk
      End While
   End If
Next i
'------------------------
```

でも同じ処理が可能です。こちらの方がプログラムとしては少し長くなりますが，w_k(i) = 0 の判定処理が ${}_mP_k$ 回と，少なくなるので，効率は良くなります。これらの処理をした後

```
'------------------------
For i=1 To k
   NT(prime(i))=0
Next i
'------------------------
```

と設定をします。

この方法を使って，w_4 を使った車輪のプログラムを書いてみましょう。

$mP_4 = 2 * 3 * 5 * 7 = 210$ ですから，まず，$w_4(0)$～$w_4(209)$ のデータを作成する必要があります。これは上述の「車輪配列 wk 作成擬似コード」利用します。また，w_0～w_3 篩の代替処理として，上述のものを使います。

ここでは，少し大きな素数を求めることとして，MaxN = 10000000 = 10^7 としてあります。

プログラム **3.1(車輪篩)**  [*20]

```
'-----------------------------------
' MaxN 以下の素数表の作成
' 4 番目までの車輪を利用
'-----------------------------------
Const MaxN=10000000
Dim NT(MaxN)
Const mP4=210 :'2*3*5*7
Dim w4(mP4)
Dim prime(4)
prime(1)=2: prime(2)=3:prime(3)=5:prime(4)=7

' w4 の設定
pt = 1
For i=2 To mP4-1
   If Gcd(i,mP4)=1 then
      w4(pt)=i-pt
      pt=i
   End If
Next i
w4(mP4-1)=2

'  w0～w3 篩の代替処理
For i=0 To mP4-1
   If w4(i)=0 then
      j=i
      While j<=MaxN
         NT(j)=1
         j=j+mP4
      End While
   End If
Next i
For i=1 To 4
   NT(prime(i))=0
Next i

' w4 車輪による篩
p=11
While p*p<=MaxN
   i=p
   While i<= Int(MaxN/p)
      NT(i*p)=1
      i =  i + w4(i mod mP4)
   End While
   p=p+2
   While NT(p)=1:p=p+2: End While
End While

'結果の表示
For i=2 To MaxN
'   必要なら 素数を表示の処理を記述
   If NT(i)=0  Then Counter= Counter +1
Next i
Print "the Number of Primes less than"; MaxN;" is ";Counter;"."
End
'-----------------------------------
```

[*20] このプログラムを私の環境で実行した場合，' w4 の設定：1 秒未満，' w0～w3 篩の代替処理：3 秒，' w4 車輪による篩：2 秒，' 結果の表示：4 秒掛かりました。

3.3 車輪の効用

前節では，車輪を使うと，篩の高速化が可能になることを見てきました。ここでは，この車輪を使うことによる篩の効率化について調べてみましょう。

車輪 $\mathtt{w_k}$ は，篩う乗数を $\mathtt{mP_k}$ と互いに素な数に限ることで，篩う個数を減らし，効率化を図るものでした。$k=1$ の場合は，2 と互いに素な数を乗数とすることで，乗数の候補を半分にすることができます。また，$k=2$ の場合は，6 と互いに素な数を乗数とすることで，乗数の候補を 1/3 にすることができます。

一般に，$\mathtt{w_k}$ については，$\mathtt{mP_k}$ 以下で，$\mathtt{mP_k}$ と互いに素な数の個数と $\mathtt{mP_k}$ との比が効率化の指標になります。

正整数 n に対して，n と互いに素な数の表を作ることや，その個数を数えることは，古くから重要な問題として研究されてきました。

オイラー (L. Euler 1707-1783) は正整数 n に対して，n 以下の n と互いに素な正整数の個数の計算を行い，次を示しました。

命題 3.3 (オイラーの ϕ 関数)**.** 正整数 n に対して，$1,2,\ldots,n$ の中で n と互いに素な数の個数を $\phi(n)$ と表す[*21]。このとき，n の素因数分解を $e_i>0$ に対して

$$n = p_1^{e_1}\cdot p_2^{e_2}\cdots p_r^{e_r}$$

とすると[*22]，

$$\phi(n) = n\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_r}\right)$$

となる[*23]。

例 3.3. 上の命題を使うと，車輪を使った篩の効率が比較できます。以下は $k=1,\ldots 9$ までの比較です。

k	1	2	3	4	5	6	7	8	9
$\mathtt{mP_k}$	2	6	30	210	2310	30030	510510	9699690	223092870
$\phi(\mathtt{mP_k})$	1	2	8	48	480	5760	92160	1658880	36495360
$\phi(\mathtt{mP_k})/\mathtt{mP_k}$	0.5	0.333	0.267	0.229	0.208	0.192	0.181	0.171	0.164
$\mathtt{mP_k}/\phi(\mathtt{mP_k})$	2	3	3.75	4.375	4.8125	5.2135	5.5393	5.8471	6.1129

$\phi(\mathtt{mP_k})/\mathtt{mP_k}$ が基本篩に対する $\mathtt{w_k}$ 篩の計算量で，その逆数 $\mathtt{mP_k}/\phi(\mathtt{mP_k})$ が効率になります。車輪が大きくなればなるほど効率は良くなりますが，車輪を使った篩の計算速度はそのまま出るわけではありません。

実際，$\mathtt{w_k}$ 車輪よる篩プログラムの実行は，$\mathtt{w_k}$ の設定，$\mathtt{w_0}$〜$\mathtt{w_{k-1}}$ 篩の代替処理を伴います。これらは k が大きいと，時間のかかる処理ですから，篩う全体の大きさ MaxN との関係で，実際の効率は k の大きさに比例するわけではありません。

[*21] この記号はガウスによるものです。

[*22] ここでの p_i は素数を小さいほうから数えて，i 番目の素数を表すものではありません。

[*23] この命題の証明はここでは省略します。証明は，どの初等整数論の教科書にも載っているでしょう。また，本サイト www.tbasic.org のプログラミングの背景の「オイラーの定理，オイラー関数」の項にもあります。

そこで実際に，プログラムを動かして比較してみましょう。MaxN として，10^7，$5 \cdot 10^7$，10^8 の場合を基本篩，改良篩，車輪篩（$k = 2, 3, \ldots, 9$）について計算した結果を表にしたものが次です[*24]。

MaxN	基本篩	改良篩	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$
10^7	12秒	9秒	7秒	5秒	5秒	5秒	5秒	5秒	19秒	—
$5 \cdot 10^7$	1分	47秒	29秒	26秒	24秒	23秒	23秒	22秒	33秒	—
10^8	2分3秒	1分34秒	59秒	51秒	48秒	—	—	—	—	—

[*25]

この結果を見ると，車輪の利用により確かに高速化されることが分かります。また，同じ k であれば，MaxN が大きくなるほど効率が良くなることも分かります。しかし一方，篩う数 MaxN の大きさに比べて k が大き過ぎると効率が下がることもあります。これは k が大きくなると，車輪の大きさも急速に大きくなり，その初期設定にかかる時間も増え，それに比べて車輪の利用による効率化は僅かな上昇になるからです。その結果ここでの $\mathtt{MaxN} = 10^8$ 程度の計算では，$k = 4$ が適していることが分かります。

使用できる配列変数が豊富で，非常に大きな MaxN についての計算であれば，$k = 5$ 以上の k が適していることもあるでしょう。

[*24] 実際の時間は実行環境によって異なります。この計算結果は私の環境 (エントリーと思われるデスクトップマシン) でのものです。ですから，結果は，時間の比較の目安と考えてください。

[*25] ここで – は配列変数不足で計算できないことを表しています。

4 篩の重複を避ける

4.1 車輪法による乗数集合の精査

車輪を使うと，篩う乗数の重複をある程度防ぎ，篩の効率化が可能でした。これは車輪が大きくなればなるほど効率化が進むことを意味します。しかし，どれほど大きくしても完全に重複を防ぐことはできません。そこでここでは，完全に重複を防ぐ方法を考えてみましょう。

車輪を利用した篩で見たように，$\mathtt{w_k}$ での篩法は，p_{k+1} で篩う場合，乗数の重複はありませんでした。しかし同じ $\mathtt{w_k}$ での篩法を使うと，次の p_{k+2} では乗数の重複が起こることがあります。しかし，$\mathtt{w_{k+1}}$ での篩法を使うと，これを避けることができます。

このことを見ると，車輪による篩を $\mathtt{w_0}$,$\mathtt{w_1}$,$\mathtt{w_2}$,... と必要なだけ順次続けて行けば乗数の重複は起きないことになります。

しかし，車輪配列 $\mathtt{w_k}$ の大きさは急激に大きくなり，MaxN に対して，十分な車輪を予め用意することは現実的ではありません。

例えば，MaxN = 1000 の場合でも，$\sqrt{10000} = 31.\cdots$ ですから，$2, 3, 5, 7, \ldots, 31$ と 11 個の素数で篩う必要があります。車輪を順次大きくして行って，31 まで篩うとすれば，$p_{10} = 29$ として，$\mathtt{w_{10}}$ まで車輪が必要になります。しかし，$\mathtt{mP_{10}} = 6469693230$ ですから，$k = 10$ に対して，配列変数 $\mathtt{w_k}$ を用意するのは現実的ではありません。一方，MaxN = 1000 の場合は，$p_{11} = 31$ まで篩う必要がありますが，実際に篩う際，$\mathtt{w_{10}}$ のすべての情報を使っているわけではありません。今の場合，乗数は $p = 31, \ldots, \left[\dfrac{1000}{31}\right] = 31$ で，車輪 $\mathtt{w_{10}}$ の情報で使うのは $p = 31$ が最初の乗数であるということだけです。

そこで，車輪を順次大きくするのではなく，予め固定した比較的大きな配列を用意して，それに車輪と同様な情報を格納することを考えてみましょう。ここでは簡単のため，MaxN の大きさの配列変数を考えます[*26]。車輪配列に格納されていた情報は，次の $\mathtt{mP_k}$ と互いに素な数との差でしたが，ここでは，差ではなく，次の数 $i + \mathtt{w_k}(i \bmod \mathtt{mP_k})$ を格納することにします。

この「次の数を格納する」という意味から，$\mathtt{w_k}$ の代わりに $\mathtt{nextS_k}$ という配列を考えましょう。即ち次の定義をします。

定義 4.1. $\mathtt{nextS_k}(1)$〜$\mathtt{nextS_k}(\mathtt{MaxN})$ を整数の配列変数とし，$i = 1, 2, 3, \ldots, \mathtt{MaxN}$ に対して，

$\gcd(i, \mathtt{mP_k}) > 1$	のとき	$\mathtt{nextS_k}(i)$の値は不定とし，$\mathtt{nextS_k}(i)$ はiでは定義されないという。
$\gcd(i, \mathtt{mP_k}) = 1$	ならば	$\mathtt{nextS_k}(i)$の値はmPkと互いに素な i の次の数。ただし， この数がMaxNより大きい場合は，特に値は定めず，過大であるという。

とする[*27]。

命題 3.1 より，$\mathtt{mP_k}$ と互いに素な i ($< \mathtt{MaxN}$) に対して，$\mathtt{nextS_k}(i) = i + \mathtt{w_k}(i \bmod \mathtt{mP_k})$ となり，$\mathtt{nextS_k}$ は車輪配列から計算できます。即ち，$\mathtt{nextS_k}$ は車輪配列の情報を $i = 1, 2, 3, \ldots, \mathtt{MaxN}$ に拡張したことになります。

[*26] 工夫をすれば，煩雑になりますがもっと少ない配列変数で同様なことが可能です。

[*27] $\gcd(i, \mathtt{mP_k}) > 1$ のとき不定とは，実際にはこの場合の値は使用されず，どのような値でも良いという意味ですが，$\mathtt{nextS_k}(i)$ を関数と見たときには，この場合は定義されないと考えます。$\gcd(i, \mathtt{mP_k}) = 1$ となる，i に対しては $\mathtt{nextS_k}(i)$ が関数として定義され，それらの i は，$i > 1$ なら，$\gcd(j, \mathtt{mP_k}) = 1$, $j > i$ となる最小の j に対して，$j = \mathtt{nextS_k}(i)$ として表されます。
また，過大であるとは，この j が，$j > \mathtt{MaxN}$ であり，実際には使われない大きな数であること意味します。プログラムの中で使う場合には，実際の値を割り当てる必要がありますが，この場合は，MaxN より大きな数，例えば $\mathtt{MaxN} + 1$ を使うことが可能です。

例を挙げましょう。

例 **4.1.** $k = 0, 1, 2, 3$ に対して，$\mathtt{nextS_k}(i)$ を $i = 1, 2, \ldots, 30$ まで表にしてみると次のようになる。ここでは $\mathtt{MaxN} = 30$ としてみます[*28]。

(1) $k = 0$ のとき，$\mathtt{nextS_k}(i) = i + 1$ となる。実際，

i	0
$\mathtt{w_0}(i)$	1

より，または，$\mathtt{mP_0} = 1$ より，定義から，すべての i に対して，次を得る。

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathtt{nextS_0}(i)$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\mathtt{nextS_0}(i)$	17	18	19	20	21	22	23	24	25	26	27	28	29	30	+

ここで，+ は過大である数を表します。

(2) $k = 1$ のとき，

i	0	1
$\mathtt{w_1}(i)$	0	2

より，i が偶数のとき，$\mathtt{nextS_1}(i)$：不定で，奇数の時，$\mathtt{nextS_1}(i) = i + 2$ となる。即ち，次を得る。

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathtt{nextS_1}(i)$	3	−	5	−	7	−	9	−	11	−	13	−	15	−	17
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\mathtt{nextS_1}(i)$	−	19	−	21	−	23	−	25	−	27	−	29	−	+	−

ここで，− は不定の数（定義されない数）を表します。これらの数（− や +）は乗数には使われないものです。以下の例で見るように，k が大きくなるにつれて，不定数が増えて，乗数が減って行くことが分かります。

(3) $k = 2$ のとき，$\mathtt{mP_2} = 2 \cdot 3 = 6$ で

i	0	1	2	3	4	5
$\mathtt{w2}(i)$	0	4	0	0	0	2

より，次を得る。

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathtt{nextS_2}(i)$	5	−	−	−	7	−	11	−	−	−	13	−	17	−	−
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\mathtt{nextS_2}(i)$	−	19	−	23	−	−	−	25	−	29	−	−	−	+	−

[*28] 実際の MaxN はもっと大きな数になりますが，境界値の状況を確認するために小さな数としました。

(4) $k = 3$ のとき，$\mathtt{mP}_3 = 2 \cdot 3 \cdot 5 = 30$ で，$\mathtt{w}_3$ の表（例 3.2）より，次を得る。

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathtt{nextS}_3(i)$	7	−	−	−	−	−	11	−	−	−	13	−	17	−	−

i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\mathtt{nextS}_3(i)$	−	19	−	23	−	−	−	29	−	−	−	−	−	+	−

これらの例からも分かるように，i で $\mathtt{nextS_k}(i)$ が定義されれば，定義から，i も $\mathtt{nextS_k}(i)$ も $\mathtt{mP_k}$ と互いに素なので，$j = \mathtt{nextS_k}(i)$ に対して，$j < \mathtt{MaxN}$ なら，j でも，$\mathtt{nextS_k}(j)$ が定義されます。$i = a_1 = 1$ から始めて，$a_{i+1} = \mathtt{nextS_k}(a_i)$ で $a_1, a_2, \ldots$ で定められる数は MaxN になるまで，$\mathtt{nextS_k}(i)$ が定義される数 i 全てを渡ります。

また，$\mathtt{nextS}_0(1) = 2$，$\mathtt{nextS}_1(1) = 3$，$\mathtt{nextS}_2(1) = 5$，$\mathtt{nextS}_3(1) = 7$ となっていて，$\mathtt{nextS_k}(1) = p_{k+1}$ が成立しています。このことは一般に成立します。即ち，次が成立します。

命題 4.1. $p = \mathtt{nextS_k}(1)$ は素数 p_{k+1} である。

証明. p が素数でなければ，p を割り切る素数 q $(q < p)$ が存在する。q は $\mathtt{mP_k}$ と互いに素だから，$1 < q < p$ より，p が 1 の次にある $\mathtt{mP_k}$ と互いに素な数であることに矛盾する。

$p = \mathtt{nextS_k}(1)$ は $p_1, \ldots p_k$ と異なる最小の素数だから $p = p_{k+1}$ である。 □

$\mathtt{nextS_k}$ から $\mathtt{nextS_{k+1}}$ への変化の状況を見ると，p_{k+1} の倍数を順次飛ばすように「次の数」を指定しなおして，ショートカットを作成していることが分かります。即ち，$\mathtt{nextS_{k+1}}$ では，$f \cdot p_{k+1}$ の「一つ前の数」の，「次の数」として $f \cdot p_{k+1}$ の（$\mathtt{nextS_k}$ での）「次の数」を指定します。例えば，$k = 2$ の場合，$p_2 = 5$ で 5 の倍数を飛ばしますが，5 を飛ばすとするとき $\mathtt{nextS}_2$ では 5 の次は 7，前は 1 なので，$\mathtt{nextS}_3$ の 1 の次は 7 になっています。

このように，$\mathtt{nextS_k}$ から $\mathtt{nextS_{k+1}}$ を構成するには，次の数だけでなく，一つ前の数の情報も必要になります。

そこで，一つ前の数 $\mathtt{prevS_k}$，即ち，$\mathtt{nextS_k}$ の逆を次のように定義します。

定義 4.2. $\mathtt{prevS_k}(1)$〜$\mathtt{prevS_k}(\mathtt{MaxN})$ を整数の配列変数とし，$i = 1, 2, 3, \ldots, \mathtt{MaxN}$ に対して，

$\gcd(i, \mathtt{mP_k}) > 1$　のとき　$\mathtt{prevS_k}(i)$の値は不定とし，$\mathtt{prevS_k}(i)$はiでは定義されないという。
$\gcd(i, \mathtt{mP_k}) = 1$　のとき　$i > 1$ならば，$\mathtt{prevS_k}(i)$は$\mathtt{mP_k}$と互いに素なiの一つ前の整数とする。

とする[*29]。

$\mathtt{prevS_k}(i)$ についても，$\mathtt{nextS_k}(i)$ と同様に $k = 0, 1, 2, 3$ に対して，表を作ってみると次のようになります。

例 4.2. (1) $k = 0$ のとき，

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathtt{prevS}_0(i)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\mathtt{prevS}_0(i)$	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

[*29] $k > 0$ ならば，$\mathtt{prevS_k}(1) = -1$ となります。

(2) $k = 1$ のとき，

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathtt{prevS}_1(i)$	−1	−	1	−	3	−	5	−	7	−	9	−	11	−	13
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\mathtt{prevS}_1(i)$	−	15	−	17	−	19	−	21	−	23	−	25	−	27	−

(3) $k = 2$ のとき。

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathtt{prevS}_2(i)$	−1	−	−	−	1	−	5	−	−	−	7	−	11	−	−
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\mathtt{prevS}_2(i)$	−	13	−	17	−	−	−	19	−	23	−	−	−	25	−

(4) $k = 3$ のとき。

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathtt{prevS}_3(i)$	−1	−	−	−	−	−	1	−	−	−	7	−	11	−	−
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\mathtt{prevS}_3(i)$	−	13	−	17	−	−	−	19	−	−	−	−	−	23	−

このように $\mathtt{prevS}_k(i)$ を定めると，$f \geqq 1$ に対し[*30]，$f \cdot p$ を飛ばし，$\mathtt{nextS}_{k+1}$ へのショートカットを作る処理は，$f \cdot p \leqq \mathtt{MaxN}$ ならば，

$$\mathtt{nextS}_{k+1}(\mathtt{prevS}_k(f \cdot p)) = \mathtt{nextS}_k(f \cdot p)$$

となります[*31]。また，$\mathtt{prevS}_{k+1}$ へのショートカットを作る処理は，$\mathtt{nextS}_k(f \cdot p) \leqq \mathtt{MaxN}$ ならば，

$$\mathtt{prevS}_{k+1}(\mathtt{nextS}_k(f \cdot p)) = \mathtt{prevS}_k(f \cdot p)$$

となります。また，$\mathtt{nextS}_k(f \cdot p) > \mathtt{MaxN}$，即ち，$\mathtt{nextS}_k(f \cdot p)$ が過大ならば，処理対象が範囲外にあり，$\mathtt{prevS}_{k+1}$ に対する処理は必要ありません。

$\mathtt{prevS}_k$ は $\mathtt{nextS}_k$ を使うと，$\mathtt{nextS}_k(i) \leqq \mathtt{MaxN}$ のとき，$\mathtt{prevS}_k(\mathtt{nextS}_k(i)) = i$ によって計算できます。
更に，$\mathtt{prevS}_k$ と $\mathtt{nextS}_k$ は $\mathtt{prevS}_0$ と $\mathtt{nextS}_0$ から，次のように，順次構成できます。

[*30] $f > 1$ ならば，$f = \mathtt{nextS}_k(g)$ の形になります。
[*31] $\mathtt{nextS}_k(f \cdot p)$ が過大の場合は，$\mathtt{nextS}_{k+1}(\mathtt{prevS}_k(f \cdot p))$ は過大になります。

命題 4.2. $\mathtt{nextS_k}$ と $\mathtt{prevS_k}$ が与えられたとき，$\mathtt{nextS_{k+1}}$ と $\mathtt{prevS_{k+1}}$ は次のようにして，構成される。

(1)（初期化）$\mathtt{nextS_{k+1}}$ と $\mathtt{prevS_{k+1}}$ の内容をそれぞれ，$\mathtt{nextS_k}$ と $\mathtt{prevS_k}$ から複写する。即ち，

```
For i = 1 to MaxN
        nextS_{k+1}(i) = nextS_k(i)
        prevS_{k+1}(i) = prevS_k(i)
Next i
```

とする。

(2)（ショートカットの作成）

```
p = nextS_k(1)
f = 1
While f * p ≦ MaxN
        prevS_{k+1}(nextS_k(f * p)) = prevS_k(f * p)
        If prevS_k(f * p) ≦ MaxN Then nextS_{k+1}(prevS_k(f * p)) = nextS_k(f * p)
        f = nextS_k(f)
End While
```

証明. $k = 0$ のとき，定義から，$1 \leqq i < \mathtt{MaxN}$ のとき，$\mathtt{nextS_0}(i) = i + 1$ で $\mathtt{nextS_0}(\mathtt{MaxN}) = \mathtt{MaxN}$ であり，$1 < i \leqq \mathtt{MaxN}$ のとき，$\mathtt{prevS_0}(i) = i - 1$ で $\mathtt{prevS_0}(1) = 1$ である。

そこで，$\mathtt{nextS_k}$ と $\mathtt{prevS_k}$ が定義通り構成されていると仮定する。このとき，上の手続きにより $\mathtt{nextS_{k+1}}$ と $\mathtt{prevS_{k+1}}$ が定義通り構成されることを示す。

まず，(1) の初期化により，$\mathtt{nextS_{k+1}}$ と $\mathtt{prevS_{k+1}}$ は $\mathtt{nextS_k}$ と $\mathtt{prevS_k}$ と同じになる。

次に，(2) の処理を実行する。$p = \mathtt{nextS_k}(1)$ は命題 4.1 より，$p = p_{k+1}$ である。$\gcd(i, \mathtt{mP_k}) = 1$ でかつ，$\gcd(i, \mathtt{mP_{k+1}}) > 1$ となる i は，p の倍数で，$i = f \cdot p$ の形になる。ここで，$\gcd(f, \mathtt{mP_k}) = 1$ より，$f = 1$ または，$f = \mathtt{nextS_k}(g)$ の形に表される。従って，この i は，(2) の処理で飛ばされる。逆に (2) の処理で飛ばされる i は p の倍数である。故に，(2) の処理で飛ばされる i は $\gcd(i, \mathtt{mP_k}) = 1$ でかつ，$\gcd(i, \mathtt{mP_{k+1}}) > 1$ となる i に他ならない。従って，(2) の処理後，$\mathtt{nextS_{k+1}}$ 及び，$\mathtt{prevS_{k+1}}$ が定義の通りのものになる。 □

上の命題 4.2 によれば，$\mathtt{nextS_0}$ と $\mathtt{prevS_0}$ から始めて，順次 $\mathtt{nextS_k}$ と $\mathtt{prevS_k}$ を計算することができます。そして，$\mathtt{nextS_k}$ を使えば，p_{k+1} での篩が重複なく可能になります。ですから，必要なだけ，$\mathtt{nextS_k}$ と $\mathtt{prevS_k}$ を予め用意しておけば，それらを使って，重複なく篩が可能となり，効率的に計算ができるとも推測されます。このアイデアに基づいて，プログラムを書くとすると次のような擬似コードが得られます[*32]。

[*32] この擬似コードでは，$\mathtt{nextS_k}$ の k を動かしていますが，このような処理を tbasic で行うには，2 次元配列を使って，`Dim nextS(10, 1000)` のような宣言をして，`nextS(k, i)` のように使う必要があります。

```
'------------------------------
' 車輪配列を順次使った篩の擬似コード
'------------------------------

k = 0からnextS_kとprevS_kの確保
k = 0
p = nextS_k(1)
While p^2 <= MaxN
      i = p
      While i <= Int(MaxN/p)
            NT(i * p) = 1
            i = nextS_k(i)
      End While
      nextS_kとprevS_kから,nextS_{k+1}とprevS_{k+1}を構成
      k = k + 1
      p = nextS_k(1)
End While

'------------------------------
```

しかし，このアイデアにはいくつか問題があります。

(1) $\mathtt{nextS_k}$ と $\mathtt{prevS_k}$ は多くの配列変数を使い，それをいくつも用意するのは現実的ではない。
(2) 上の命題による $\mathtt{nextS_k}$ と $\mathtt{prevS_k}$ 構成処理にはかなりの時間が掛かる。

例えば，$\mathtt{MaxN} = 10000$ の場合でも，$k = 24$ までの $\mathtt{nextS_k}$ と $\mathtt{prevS_k}$ が必要になり，$2 * 10000 * 24 = 480000$ 個の配列変数が必要になります。また，命題 4.2(1) での初期化もかなりの時間が掛かります。

しかし，これらの問題は，$\mathtt{nextS_k}$ と $\mathtt{prevS_k}$ を全ての k に対して共通的に使うことができれば解決できると推測されます。この考え方について次節で考えてみましょう。

4.2 リストに篩をかける

$\mathtt{nextS_k}$ と $\mathtt{prevS_k}$ を全ての k に対して共通的に使うというアイデアは配列変数を効率的に利用でき，有効です。しかし，それらは，nextS と prevS の配列それぞれについて，並行的に k を動かす処理が複雑に見えます。

しかし，それらを篩の対象である NT 配列とまとめて考える，即ち，NT 配列を nextS と prevS の配列とまとめて双方向リスト (doubly-linked list) として扱うと，それらの関係が明確になり，分かりやすくなります。そして篩をこれらのリストに対して行います[*33][*34]。

[考え方]

SList を $2, 3, 4, \ldots, \mathtt{MaxN}$ を元とする双方向リストとします。このリストに篩を施します。即ち，順次合成数を除いて行き，最終的に素数のみ含むリストにします。このリストには特にデータを格納することはなく，リストに対する処理はリストにある元 t の削除のみです。この削除の処理を $\mathtt{deleteS}(t)$ と表すことにします。リストに含まれる元の大小関係は対応する数の大小関係とします。リストにある各元 t に対して，リストにある次の元を示す $\mathtt{nextS}(t)$ と一つ前の元を示す $\mathtt{prevS}(t)$ が与えられていて，リストの元を削除する都

[*33] 双方向リストはよく知られたデータ構造ですから，インターネット上でも多くの参考文献が入手できます。tbasic.org「プログラミングの背景」旧コンテンツにもその解説があります。
[*34] 以下に使われる双方向リストは単純なものですから，リストについての知識は必ずしも必要ではありません。

度，対応するこれらの値が更新されます。nextS(t) と prevS(t) の初期値はそれぞれ，$t = 2, 3, 4, \ldots,$ MaxN に対して，nextS(t) = $t + 1$，prevS(t) = $t - 1$ とします[*35]。このリストの最初の元は 2 で prevS(2) = 1 で，これは，2 が最初の元であることを示します[*36]。また，初期値では，nextS(MaxN) = MaxN + 1 ですが，一般に，nextS(t) = MaxN + 1 となるとき，t は最後の元であることを示します。

このリストを篩っていく状況を例として挙げます。

例 **4.3.** MaxN = 30 とします。この場合，$\sqrt{\text{MaxN}} < 7$ より，$p = 2, 3, 5$ で篩えば終了です[*37]。

(1) 初期設定のとき。

SList : i		2	3	4	5	6	7	8	9	10	11	12	13	14	15
nextS(i)		3	4	5	6	7	8	9	10	11	12	13	14	15	16
prevS(i)		1	2	3	4	5	6	7	8	9	10	11	12	13	14
SList : i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
nextS(i)	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
prevS(i)	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

(2) $p = 2$ で篩った（4 以上の偶数をリストから削除した。）後。

SList : i		2	3		5		7		9		11		13		15
nextS(i)		3	5		7		9		11		13		15		17
prevS(i)		1	2		3		5		7		9		11		13
SList : i	17		19		21		23		25		27		29		
nextS(i)	19		21		23		25		27		29		31		
prevS(i)	15		17		19		21		23		25		27		

(3) $p = 3$ で篩った（9, 15, 21, 27 をリストから削除した。）後。

SList : i		2	3		5		7				11		13		
nextS(i)		3	5		7		11				13		17		
prevS(i)		1	2		3		5				7		11		
SList : i	17		19				23		25				29		
nextS(i)	19		23				25		29				31		
prevS(i)	13		17				19		23				25		

(4) $p = 5$ で篩った（25 をリストから削除した。）後。

SList : i		2	3		5		7				11		13		
nextS(i)		3	5		7		11				13		17		
prevS(i)		1	2		3		5				7		11		
SList : i	17		19				23						29		
nextS(i)	19		23				29						31		
prevS(i)	13		17				19						23		

この処理の結果，リストに残った 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 が 30 以下の素数の全体になります。

[*35] nextS(MaxN) = MaxN + 1，prevS(2) = 1 はリスト外の元ですが，これらの値は元としては使用しないとします。

[*36] このリストでは，2 を取り除くことがないので，2 は常に最初の元になります。

[*37] 以下で，nextS(i) の値として 31 が出てきますが，これは対応する i がリストの最後の元であること示しています。

車輪の拡大での処理と同様に，双方向リスト SList で，素数 p と $p \leq i$ なる i に対して，元 $i \cdot p \leq \mathtt{MaxN}$ を削除 $\mathtt{deleteS}(i \cdot p)$ 処理は

$\mathtt{nextS}(\mathtt{prevS}(i \cdot p)) = \mathtt{nextS}(i \cdot p)$
$\mathtt{if\ nextS}(i \cdot p) \leqq \mathtt{MaxN\ Then\ prevS}(\mathtt{nextS}(i \cdot p)) = \mathtt{prevS}(i \cdot p)$ (∗delete prevS)

で与えられます。既に削除（delete 処理を）された元に再びこの delete 処理を行うと，不具合が起きる可能性がありますから，削除された元を判定する方法か，２度削除を行わないようにする必要があります。

リストの元は順次削除されていくため，次の元や前の元との間は離れていきますが，これらの関数 $\mathtt{nextS}(t)$ と $\mathtt{prevS}(t)$ を使うことによって高速にそれらの元が得られます。

この双方向リストを使って篩を行う擬似コードを書いてみましょう。

まず，双方向リスト SList の確保し，$\mathtt{nextS}(t)$ と $\mathtt{prevS}(t)$ の初期値を設定します[*38]。$p = 2$ から始めて順次素数 p で篩っていきます。篩う素数は $\sqrt{\mathtt{MaxN}}$ 以下です。p で篩うには，SList に残っている $p \leqq i \leqq \mathtt{Int}(\mathtt{MaxN}/\mathtt{p})$ となる i を動かし，$i \cdot p$ を篩います。その際，$i \cdot p$ はリストから削除します。p で篩い終わったとき，SList に残っている p の次の元は素数ですから，次に $\mathtt{nextS}(p)$ で篩います。これを $\sqrt{\mathtt{MaxN}}$ 以下の範囲で行います。

以上の考えを上述の「車輪配列を順次使った篩の擬似コード」と同様にして，擬似コードとして表すと次のようになります。

[擬似コード]

```
'------------------------------
' 双方向リストを使った篩の擬似コード
'------------------------------

双方向リストSListの確保
p = 2
While p^2 <= MaxN
      i = p
      While i <= Int(MaxN/p)
            Delete(i * p)
            i = nextS(i)
      End While
      p = nextS(p)
End While

'------------------------------
```

この擬似コードを検証してみましょう。

篩は $p = 2$ から始めます。一般に，p で篩うには，$\sqrt{\mathtt{MaxN}}$ 以下の p で十分です。また p より大きい p の倍数をリストから削除していきますから，整数 $i > 1$ に対して $i \cdot p$ の形の数を削除します。ここで，i は基本篩法の説明にもあったように，p が最初でそれ以上の数で十分です。また，$i \cdot p$ は MaxN 以下の範囲ですから，i は $\mathtt{Int}(\mathtt{MaxN}/\mathtt{p})$ 以下になります。即ち，

$$p \leq i \leq \mathtt{Int}\left(\frac{\mathtt{MaxN}}{p}\right) \qquad \text{(*ifactor)}$$

の範囲の i になります。ここで，p 未満の素因数をもつ i は，既に篩われていますから，i は p 以上の素因数を持つ数です。即ち，i は (∗ifactor) の範囲にある p 以上の素因数を持つ数，全てに渡ります。

[*38] SList の具体的実装法は以下で説明します。

ところで，i = nextS(i) で与えられる i は，p 未満の篩を実行が終了した時点では，p 以上の素因数を持つ数全てが残っていますが，p での篩を実行していく中で $i \cdot p$ の元が削除され，i は p 以上の素因数を持つ数全てに渡るとは言えません。

例えば，$i = p$ に対して，$p \cdot i$ を篩うと，その時点で p^2 が削除され，それ以降の i = nextS(i) の中には p^2 は現われません。即ち，上の擬似コードでは $p \cdot p^2$ が篩われず，篩い残しが生じます！

これは，乗数 i を小さいほうから大きい方に向かって取っていくに対して，篩によって $p \cdot i$ を篩うことから生じます。ここで篩は $p \cdot i$ の形の数を篩わなければなりませんが，乗数 i はどのような取り方をしても全体を渡れば良いだけです。ですから，乗数 i を大きいほうから小さい方へと取っていけばこのことを避けることができます。nextS(i) を使って $\frac{\mathrm{MaxN}}{\mathrm{p}}$ を超えない最大の整数 i を求めて，それを出発点とし，prevS(i) を使って p まで，小さい方に向かって取って行けばよいことになります。

この修正を上の擬似コードに施すと，次が得られます。

```
'------------------------------
' 修正された
' 双方向リストを使った篩の擬似コード
'------------------------------

双方向リストSListの確保
p = 2
While p^2 <= MaxN
      i = p
      While nextS(i) * p <= MaxN
            i = nextS(i)
      End While
      While p <= i
            Delete(i * p)
            i = prevS(i)
      End While
      p = nextS(p)
Wend

'------------------------------
```

4.3 リスト篩

この擬似コードを使って実際にプログラムを書いてみましょう。まず，SList ですが，普通の構成法だと，NT, NextS, PrevS と 3 種の MaxN 個の配列を用意して，それぞれ，データ，次の元，前の元の情報を格納します。しかし，既に注意したように，このリストは実際にはデータを格納せず，元があるかないかを判定できれば良いだけです。ですから，その構成に，配列 NT は使わないことにします。これにより使用する配列が $\frac{2}{3}$ になります。

また，prevS の配列宣言を MaxN ではなく，1 だけ増やし，MaxN+1 にします。これにより，(*delete prevS) での if 文を省略することができ，多少の高速化が図れます[*39]。

プログラム **4.1(**リスト篩法**)**

```
'------------------------------
' リスト篩法
'------------------------------
Public Const MaxN=10000000
Public Const MaxND=10000001
Public NextS(MaxN), PrevS(MaxND)

Print Time$
For i=2 To MaxN
      NextS(i) = i + 1
      PrevS(i) = i - 1
Next i

p=2
Print Time$
While p^2 <= MaxN
   i = p
   While NextS(i)*p <= MaxN
      i=NextS(i)
   End While
   Do
     PrevS(NextS(i*p)) = PrevS(i*p)
     NextS(PrevS(i*p)) = NextS(i*p)
      i = PrevS(i)
   Loop Until i < p
   p = NextS(p)
End While
Print Time$
Counter = 1
p=2
While p =< MaxN
   p=NextS(p)
   If p<MaxN  Then Counter= Counter +1
End While
 Print "The number of primes less than"; MaxN; " is ";Counter
End
'------------------------------
```

[*39] prevS(MaxN + 1) は，(*delete prevS) での限られた場合のみアクセスされ，篩終了後は，MaxN 以下最大の素数が格納されます。

この篩法による計算速度を調べてみましょう。MaxN = 10^7, $2\cdot10^7$, $3\cdot10^7$, $4\cdot10^7$, $5\cdot10^7$ について実行した結果は以下の通りです[*40]。

MaxN	10^7	$2\cdot10^7$	$3\cdot10^7$	$4\cdot10^7$	$5\cdot10^7$	$6\cdot10^7$
初期化時間	7秒	15秒	21秒	28秒	35秒	–
篩時間	9秒	20秒	29秒	40秒	49秒	–
合計時間	16秒	34秒	50秒	1分7秒	1分24秒	–

結果を見ると残念ながら，計算速度は余り速くありません。初期化を除いた，篩実行時間に限っても，車輪法に比べて遅い結果になっています。これは，リスト篩法は篩回数は少なくなっていますが，その一回にかかる処理時間が他の方法よりもかかるためです。

しかし，リスト篩法は同じものを２度篩うことがありませんから，全体の篩回数は MaxN 以下です。更に一回の篩にかかる時間は MaxN に関わらず，一定ですから，篩全体に必要な計算時間は，ある適当な定数 K に対して，K MaxN 以下になります。つまり，同じ環境で計算を行った場合，MaxN を 100 倍にすれば，計算時間も 100 倍程度で収まることを意味しています。上の計算結果もそのことを示しています。

これに対して，車輪篩法は k を止めて，MaxN を大きくしていくと，重複して篩う数が増えていきます。そのため，全体の計算時間は MaxN の比よりも増加していくことが推測されます。そしてこのことは実際に示されています。ですから，非常に大きな MaxN について車輪篩法とリスト篩法を比較すれば，リスト篩法の方が高速に実行される筈です。しかし，現実問題として非常に大きな MaxN での実行という状況は起きえませんから，標準的な計算機を用いた計算でリスト篩法は高速計算向きとは言えないでしょう。

[*40] MaxN = $6\cdot10^7$ はメモリー不足のため実行できませんでした。また，秒数は小数点以下四捨五入のため，初期化時間と篩時間の合計が合計時間と異なる場合があります。

5　より少ない配列変数で篩う

前節以前で説明した方法，特に車輪篩法は，篩うために必要な配列変数が確保できれば，プログラムも簡単で，効率的に動作します。ですからこの方法は大きな素数表を作成する非常に良い方法です。

しかし，場合によってはより少ない配列変数で作成したいこともあるかもしれません。

例えば，100 以下の素数は 25 個ですが，前節以前の方法ではそれを篩うために 100 の配列変数を使います。また同様に，1000，$10000 = 10^4$ 以下の素数，それぞれ，168 個，1229 個を求めるに 1000 個，$10000 = 10^4$ 個の配列変数を使います。求める個数と必要な配列の個数を比較すると，もっと少ない配列変数で素数表を作成できる方法があるのではと思います。

ここでは，この問題を考えてみましょう。

5.1　奇数のみを篩う

利用する配列変数を減らす方法としてすぐに思いつくのは，奇数のみで，篩うことです。この方法はこの文書の冒頭に挙げたニコマコスの「算術入門」での方法です。この方法ですと，半分の配列変数で篩が可能になります。言い換えると，同じ配列変数で，2 倍の素数表を作ることができます。

ここでは，このアイデアで以前の節と同じ 100 個の配列変数 $\mathtt{NT}(i)$ $(i = 1, 2, \ldots, 100)$ 使って，素数表を作ってみましょう。この場合，この方法で 200 以下の素数表を作ることができます。

200 以下の奇数の表 $j = 1, 3, 5, \ldots, 199$ を作ると次のようになります。

1	3	5	7	9	11	13	15	17	19
21	23	25	27	29	31	33	35	37	39
41	43	45	47	49	51	53	55	57	59
61	63	65	67	69	71	73	75	77	79
81	83	85	87	89	91	93	95	97	99
101	103	105	107	109	111	113	115	117	119
121	123	125	127	129	131	133	135	137	139
141	143	145	147	149	151	153	155	157	159
161	163	165	167	169	171	173	175	177	179
181	183	185	187	189	191	193	195	197	199

(表 1)

これらの数の素数性の情報を，篩によって配列変数 $\mathtt{NT}(i)$ を格納します。

この表は $i = 1,\ 2,\ 3,\ \ldots,\ 100$ に対して，$j = 2i - 1$ を順次並べたものです[*41]。配列変数 $\mathtt{N}(i)$ との対応は，i に対して，$\mathtt{N}(i)$ は，$j = 2i - 1$ の素数・合成数性を意味するものとします。

例えば，$\mathtt{N}(2)$ は 3 の素数・合成数性を示す数値，0 または 1 を保持します。

この表は，2 で既に篩われたものですから，$p = 3$ から篩を始めます。まず，3 で篩います。即ち，3 の奇数倍を取り除きます。乗数は 3 から始める奇数ですから，

$$3,\ 5,\ 7,\ 9,\ 11,\ \ldots,\ 65,\ 67,\ \ldots$$

となりますが，これは (表２) にある 3 以後の数になっています。$3 \times 3 = 9$ から始めて，3 の奇数倍を順次取って，$65 \times 3 = 195$, $67 \times 3 = 201$ ですから，65×3 が最後です。この最後の乗数 65 は，$\left[\dfrac{199}{3}\right] = 66$ 以下の最大

[*41] $i = 1$ のときは，$j = 1$ で，1 は篩の対象ではありませんから，以後，$i = 1$, $j = 1$ の場合は考察の対象から外すことにします。

の奇数です[*42]。除いた数に取り消し線 ── を書くと，次が得られます。

	3	5	7	~~9~~	11	13	~~15~~	17	19
~~21~~	23	25	~~27~~	29	31	~~33~~	35	37	~~39~~
41	43	~~45~~	47	49	~~51~~	53	55	~~57~~	59
61	~~63~~	65	67	~~69~~	71	73	~~75~~	77	79
~~81~~	83	85	~~87~~	89	91	~~93~~	95	97	~~99~~
101	103	~~105~~	107	109	~~111~~	113	115	~~117~~	119
121	~~123~~	125	127	~~129~~	131	133	~~135~~	137	139
~~141~~	143	145	~~147~~	149	151	~~153~~	155	157	~~159~~
161	163	~~165~~	167	169	~~171~~	173	175	~~177~~	179
181	~~183~~	185	187	~~189~~	191	193	~~195~~	197	199

取り除いた N(i) 項の i は，$i = (j+1)/2$ に注意して，

$$5 = (3\cdot 3+1)/2,\ 8 = (5\cdot 3+1)/2,\ \ldots,\ 98 = (65\cdot 3+1)/2$$

です。(表 2) でこれらに対応する位置の数が篩われ，取り消し線が引かれています。

次に，3 の次に取り消し線の無い数 $p = 5$ で篩います。乗数は $p = 3$ の場合と同様に 5 以後の奇数で，

$$5,\ 7,\ 9,\ 11,\ \ldots,\ 39,\ 41,\ \ldots,\ 65,\ 67,\ \ldots$$

(表２) にある 5 以後の数になっています。従って，$5\times 5 = 25$ から始めて，$7\times 5 = 35$ と続き，$39\times 5 = 195$, $41\times 5 = 205$ ですから，39×5 が最後です。これらを除いた数に取り消し線 ── を上の表に追加すると，次が得られます。

	3	5	7	~~9~~	11	13	~~15~~	17	19
~~21~~	23	~~25~~	~~27~~	29	31	~~33~~	~~35~~	37	~~39~~
41	43	~~45~~	47	49	~~51~~	53	~~55~~	~~57~~	59
61	~~63~~	~~65~~	67	~~69~~	71	73	~~75~~	77	79
~~81~~	83	~~85~~	~~87~~	89	91	~~93~~	~~95~~	97	~~99~~
101	103	~~105~~	107	109	~~111~~	113	~~115~~	~~117~~	119
121	~~123~~	~~125~~	127	~~129~~	131	133	~~135~~	137	139
~~141~~	143	~~145~~	~~147~~	149	151	~~153~~	~~155~~	157	~~159~~
161	163	~~165~~	167	169	~~171~~	173	~~175~~	~~177~~	179
181	~~183~~	~~185~~	187	~~189~~	191	193	~~195~~	197	199

以後，$p = 7,\ 13,\ \ldots$ と続きます。一般に，奇素数 p で篩うには，

$$r \leqq \left[\frac{199}{p}\right] = \left[\frac{2\cdot 100-1}{p}\right]$$

となる最大の奇数を r とすると，

$$j = p\cdot p,\ (p+2)\cdot p,\ (p+4)\cdot p,\ \ldots,\ r\cdot p \qquad \text{(篩 1)}$$

が篩われる数 j になります。実際，r の次の奇数を r' とすると，$r' = r+2$ で，

$$r \leqq \left[\frac{199}{p}\right] < \left[\frac{199}{p}\right] + 1 \leqq r'$$

[*42] 次の奇数は $\left[\frac{199}{3}+1\right]$ 以上ですから，命題 2.1 より，積は 199 を超えます。

となり，命題 2.1 より，

$$r \cdot p \leqq \left[\frac{199}{p}\right] \times p \leqq 199 < \left(\left[\frac{199}{p}\right] + 1\right) \times p \leqq r' \cdot p$$

が得られます。

ここで，篩 1 の N(i) 対応する i は，$j = 2i - 1$ より，$i = (j + 1)/2$ で，

$$\frac{p \cdot p + 1}{2},\ \frac{(p+2) \cdot p + 1}{2},\ \frac{(p+4) \cdot p + 1}{2},\ \ldots,\ \frac{r \cdot p + 1}{2}$$

となります。

篩う素数は $\sqrt{199} = 14.1\ldots$ 以下の素数ですから，3, 5, 7, 11, 13 です。

これらで篩い，取り消し線を追加すると，上の表は

	3	5	7	~~9~~	11	13	~~15~~	17	19
~~21~~	23	~~25~~	~~27~~	29	31	~~33~~	~~35~~	37	~~39~~
41	43	~~45~~	47	~~49~~	~~51~~	53	~~55~~	~~57~~	59
61	~~63~~	~~65~~	67	~~69~~	71	73	~~75~~	~~77~~	79
~~81~~	83	~~85~~	~~87~~	89	~~91~~	~~93~~	~~95~~	97	~~99~~
101	103	~~105~~	107	109	~~111~~	113	~~115~~	~~117~~	~~119~~
~~121~~	~~123~~	~~125~~	127	~~129~~	131	~~133~~	~~135~~	137	139
~~141~~	~~143~~	~~145~~	~~147~~	149	151	~~153~~	~~155~~	157	~~159~~
~~161~~	163	~~165~~	167	~~169~~	~~171~~	173	~~175~~	~~177~~	179
181	~~183~~	~~185~~	~~187~~	~~189~~	191	193	~~195~~	197	199

となります。この表から取り消し線のある数を消すと

	3	5	7		11	13		17	19
	23			29	31			37	
41	43		47			53			59
61			67		71	73			79
	83			89				97	
101	103		107	109		113			
			127		131			137	139
				149	151			157	
	163		167			173			179
181					191	193		197	199

が得られます。これらの数に 2 を加えた 46 個が 200 以下の素数になります。

以上のことをプログラムとして表しましょう。プログラム 2.1（基本篩）或いは，プログラム 2.2（改良篩）を参考にすると，は次のように書き換えられます。

プログラム **5.1**

```
'-----------------------------------
' 200 以下の素数表の作成
' 奇数を篩う
'-----------------------------------
Dim NT(100)
p=3
While p*p<=2*100-1
   For i=p To Int((2*100-1)/p) step 2
      NT((i*p+1)/2)=1
   Next i
   p=p+2
   While NT((p+1)/2)=1:p=p+2:Wend
End While
Print 2
For i=2 To 100
   If NT(i)=0  Then Print 2*i-1
Next i
End
'-----------------------------------
```

200 以下ではなく，2 * MaxN 以下の素数を求めるプログラムにも，以下のように簡単に変更できます。

プログラム **5.1'**

```
'-----------------------------------
' 2*MaxN 以下の素数表の作成
' 奇数を篩う
'-----------------------------------
Const MaxN=1000
Dim NT(MaxN)
p=3
While p*p<=2*MaxN-1
   For i=p To Int((2*MaxN-1)/p) step 2
      NT((i*p+1)/2)=1
   Next i
   p=p+2
   While NT((p+1)/2)=1:p=p+2:Wend
End While
Print 2
For i=2 To MaxN
   If NT(i)=0  Then Print 2*i-1
Next i
End
'-----------------------------------
```

これらプログラムはプログラム 2.2（改良篩）とほぼ同じですが，必要な配列変数が半分になるだけでなく，初期化の部分が必要ないので，高速化も期待できます[*43]。実際に実行して比較した結果は次の通りです。

篩う最大数	10^6	$5\cdot10^6$	10^7	$5\cdot10^7$	10^8	$2\cdot10^8$
プログラム2.2'(改良篩)	1秒	4秒	9秒	44秒	1分32秒	–
プログラム5.1'(奇数篩)	1秒	5秒	11秒	48秒	1分38秒	3分20秒

結果を見ると，奇数篩の方が少し時間がかかっていますが，実行速度は大体同じと考えられます。大きな違いは，改良篩ではできなかった $2\cdot10^8$ までの素数表が作成できることにあります。

[*43] 実際はそうでもありません。これは奇数篩の方が一回の篩の計算が少し複雑になっているためです。

5.2 2, 3, 5, . . . と互いに素な数の表

プログラム 5.1 では，奇数の表を使うことで，使用する配列変数を半分にすることができました。奇数の表は 2 で篩った残りから 2 を除いた表ですが，見方を変えると，2 と互いに素な数の表とも言えます[*44] 。

この見方を拡張すると，いくつかの数で篩った残りから，篩った素数を除いた表，即ち，篩う素数と互いに素な数の表を使うことで，より少ない配列変数で，より多くの素数を得ることができると予想されます。

例えば，2, 3 と互いに素な数を 100 個並べると[*45]，次が得られます。

	5	7	11	13	17	19	23	25	29
31	35	37	41	43	47	49	53	55	59
61	65	67	71	73	77	79	83	85	89
91	95	97	101	103	107	109	113	115	119
121	125	127	131	133	137	139	143	145	149
151	155	157	161	163	167	169	173	175	179
181	185	187	191	193	197	199	203	205	209
211	215	217	221	223	227	229	233	235	239
241	245	247	251	253	257	259	263	265	269
271	275	277	281	283	287	289	293	295	299

(表 2)

この表を 100 個の配列変数 $\mathtt{NT}(i)$ $(i = 1, 2, \ldots, 100)$ に対応させて篩を行えば，300 以下の素数の表を作ることができるでしょう。この場合，同じ配列変数で 3 倍の素数表を作ることができると言えます。

更に，2, 3, 5, 7, . . . と互いに素な数の表について篩を行うことも考えられます。

例えば，2, 3, 5, 7, 11, 13 と互いに素な数を 100 個並べると次が得られます[*46]。

1	17	19	23	29	31	37	41	43	47
53	59	61	67	71	73	79	83	89	97
101	103	107	109	113	127	131	137	139	149
151	157	163	167	173	179	181	191	193	197
199	211	223	227	229	233	239	241	251	257
263	269	271	277	281	283	289	293	307	311
313	317	323	331	337	347	349	353	359	361
367	373	379	383	389	391	397	401	409	419
421	431	433	437	439	443	449	457	461	463
467	479	487	491	493	499	503	509	521	523

(表 6)

この表を 100 個の配列変数 $\mathtt{NT}(i)$ $(i = 1, 2, \ldots, 100)$ に対応させて篩を行えば，525 以下の素数の表を作ることができるでしょう。この場合，同じ配列変数で 5 倍の素数表を作ることができるでしょう。

a と互いに素な数の表を作ることや，その個数を数えることは，古くから重要な問題として研究されてきました。オイラーの ϕ 関数については車輪の項で既に説明をしました（命題 3.3）が，再掲します。。

[*44] 一般に，整数 a, b に対して，それらの最大公約数が 1 のとき，a, b は互いに素，a は b と互いに素と言います。

[*45] 1 は除外します。

[*46] これらの表が (表 2) や (表 6) という番号なのに違和感を感じるかもしれませんが，これは互いに素とする素数の個数から由来した命名です。

命題 3.3(オイラーの ϕ 関数 (再掲))

正整数 n に対して，$1, 2, \ldots, n$ の中で n と互いに素な数の個数を $\phi(n)$ と表す。このとき，n の素因数分解を

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_r^{e_r}$$

とすると，

$$\phi(n) = n\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\cdots\left(1 - \frac{1}{p_r}\right)$$

となる

オイラーの ϕ 関数の結果を使って，上の (表 2)，(表 6) について調べてみましょう。まず，n と互いに素な整数は $\bmod n$ で決まることに注意しましょう。実際，すべての正整数 m は $m = qn + r$ $(0 \le r < n)$ の形に表されますから，m と n の最大公約数は r と n と同じになります。ですから，m が n と互いに素になる必要十分条件は r が n が互いに素になることです。

特に，kn 以下の数で n と互いに素なものの個数は $k\phi(n)$ になります。

例 5.1.

(1) $n = 2 \cdot 3 = 6$ の場合。

$$\phi(6) = 6\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right) = (2-1)(3-1) = 2$$

となる。従って，$50n = 300$ までの正整数で 6 と互いに素なものの個数は $50\phi(6) = 100$ 個となる。故に，6 と互いに素な 100 番目の正整数は $300 - 1 = 50n - 1 = 299$ である。これは (表 2) にある最後の数である。

(2) $n = 2 \cdot 3 \cdot 5 = 30$ の場合。

$$\phi(30) = 2 \cdot 3 \cdot 5\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right) = (2-1)(3-1)(5-1) = 8$$

となる。従って，$12n = 360$ までの正整数で 30 と互いに素なものの個数は $12\phi(30) = 96$ 個となる。360 以上で 30 と互いに素な正整数は $360 + 1,\ 360 + 7,\ 360 + 11,\ 360 + 13,\ 360 + 17,\ \ldots$ と続く。従って，373 以下の正整数で 30 と互いに素な 100 番目の正整数は 373 である。

上の例のように $n = 2 \cdot 3 = 6$ や $n = 2 \cdot 3 \cdot 5 = 30$ の場合，n と互いに素な 100 個の数の表を作成しないで，100 番目の n と互いに素な数を命題 3.3 から求めることができました。一方，$n = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 = 30030$ の場合は，n が 100 より大きいので，命題 3.3 から直接 (表 6) の最後の数を求めることは無理です。しかし，$\dfrac{\phi(n)}{n}$ は n と互いに素な数の割合を示す数と見られるので，その逆数 $\dfrac{30030}{\phi(30030)} = 5.21\ldots$ より[*47]，100 番目の n と互いに素な数は 521 付近の数であると推測できます。実際に計算してみると，523 が n と互いに素な 100 番目の数です。

[*47]

$$\begin{aligned}\phi(30030) &= 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right)\left(1 - \frac{1}{5}\right)\left(1 - \frac{1}{7}\right)\left(1 - \frac{1}{11}\right)\left(1 - \frac{1}{13}\right)\\ &= (2-1)(3-1)(5-1)(7-1)(11-1)(13-1)\\ &= 5760\end{aligned}$$

5.3 2, 3, 5, . . . と互いに素な数の表を篩う

プログラム 2.1 とプログラム 5.1 を比較してみると，配列変数と表との対応部分を除けば，随分と似た構成になっていることに気づきます。ですから，配列変数と表との対応を上手に処理すれば同様にプログラムを書くことができるのではと期待されます。

そこで，むしろ一般的な場合を考え，それについて分析することで，(表 2) や (表 6) の篩を実行するプログラムの作成を試みましょう。

一般に，素数 $p_1 = 2 < p_2 = 3 < p_3 < \ldots < p_n$ と p_n までのすべての素数に対して，それらの積を以前のように $\mathtt{mP_n} = p_1 p_2 \cdots p_n$ と表します。$\mathtt{mP_n}$ と互いに素な数の表に対して，篩を行うことを考えます。

$\mathtt{mP_n}$ と互いに素な数の表を

	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}	r_{29}	r_{30}
r_{31}	r_{32}	r_{33}	r_{34}	r_{35}	r_{36}	r_{37}	r_{38}	r_{39}	r_{40}
r_{41}	r_{42}	r_{43}	r_{44}	r_{45}	r_{46}	r_{47}	r_{48}	r_{49}	r_{50}
r_{51}	r_{52}	r_{53}	r_{54}	r_{55}	r_{56}	r_{57}	r_{58}	r_{59}	r_{60}
r_{61}	r_{62}	r_{63}	r_{64}	r_{65}	r_{66}	r_{67}	r_{68}	r_{69}	r_{70}
r_{71}	r_{72}	r_{73}	r_{74}	r_{75}	r_{76}	r_{77}	r_{78}	r_{79}	r_{80}
r_{81}	r_{82}	r_{83}	r_{84}	r_{85}	r_{86}	r_{87}	r_{88}	r_{89}	r_{90}
r_{91}	r_{92}	r_{93}	r_{94}	r_{95}	r_{96}	r_{97}	r_{98}	r_{99}	r_{100}
...									
.	.	.	.	.	.	.	.	.	$r_{\mathtt{MaxN}}$

(篩表 n)

とします[*48]。$r_1 = 1$ ですが，1 は篩の対象外なので，表には書いてありません。

$\mathtt{mP_n}$ と互いに素な表は，素数 $p_1 = 2 < p_2 = 3 < p_3 < \ldots < p_n$ で篩を行い，残った数の表から素数 $p_1 = 2 < p_2 = 3 < p_3 < \ldots < p_n$ を取り除いたものです。

今までの表は 100 個の数を並べたものでしたが，ここでは状況を一般的にするために MaxN 個の数を並べることにし，この表に篩をかけることを考えます。

説明を分かりやすくするために，必要な関数をいくつか定義しましょう。

定義 5.1.

(1) 上の r_i を関数の形式で，$\mathtt{Rp_n}(i)$ と表す[*49]。逆に，$r = \mathtt{Rp_n}(i)$ に対してその逆関数を $i = \mathtt{Ir_n}(r)$ と表す。便宜上，$i = 1$ のとき，$\mathtt{Rp_n}(1) = 1$ とする。但し $i = 1$ の場合は $r_1 = 1$ で，これは篩の対象外とする。

(2) 正整数 m に対して，m 以下の正整数で $\mathtt{mP_n}$ と互いに素な最大数を $\mathtt{Floor_n}(m)$ と表す。また，m 以上の正整数で $\mathtt{mP_n}$ と互いに素な最小数を $\mathtt{Ceil_n}(m)$ と表す。

例 5.2.

(1) 互いに素とする素数がない場合 ($n = 0$, $\mathtt{mP_0} = 1$)，即ち，(表 0) の場合，$\mathtt{Rp_0}(i) = i$，$\mathtt{Ir_0}(r) = r$，$\mathtt{Floor_0}(m) = m$，$\mathtt{Ceil_0}(m) = m$（恒等関数）である。

[*48] $n = 0$ の場合も考え，このときは，$\mathtt{mP_0} = 1$ とすることにします。そのように考えると，これは (表 0) になります。

[*49] $\mathtt{mP_n}$ と互いに素な i 番目の数 (i is relatively prime to $\mathtt{mP_n}$)

(2) 互いに素とする素数が 2 の場合 ($n = 1$, $\mathtt{mP}_1 = 2$)，即ち，(表 1) の場合，$\mathtt{Rp}_1(i) = 2i - 1$，$\mathtt{Ir}_1(s) = \dfrac{s+1}{2}$，$\mathtt{Floor}_1(m) = m - 1 + (m \mod 2)$，$\mathtt{Ceil}_1(m) = m + 1 - (m \mod 2)$ である[*50]。

この例のように，$n = 0, 1$ の場合は，$\mathtt{Rp}_\mathtt{n}(i)$，$\mathtt{Ir}_\mathtt{n}(s)$，$\mathtt{Floor}_\mathtt{n}(m)$，$\mathtt{Ceil}_\mathtt{n}(m)$ は簡単な式で計算できます。一般の n についてはこのような簡単な式はありませんが，比較的小さな n については効率的な計算法があります。これについては，次項で説明することにして，ここでは，これらの関数を使って篩法の説明をします。

篩は初めに素数を取って，それの倍数を順次取り除く方法です。そのためにまず，素数を取ります。それには次の命題を注意します。

命題 5.1. (篩表 n) において，p_n の次の素数 q_1 から始めて，いくつかの素数 $q_1, q_2, \ldots, q_k$ による篩を順次行ったとき，篩われずに残っている q_k の次の数 $r_t > 1$ は素数である。
特に，$r_2 = \mathtt{Rp}_\mathtt{n}(2)$ は素数である。

証明. r_t が合成数なら，は $p < r_t$ となる r_t の素因数 p が存在する。ここで，r_t は $p_1, p_2, p_3, \ldots, p_n$ と互いに素だから，p も $p_1, p_2, p_3, \ldots, p_n$ と互いに素になる。r_t は q_r の次の数だから，p は $q_1, q_2, \ldots, q_r$ の何れかと一致するか，それらで篩われた数である。一方，素数は篩われる数ではないから，p は $q_1, q_2, \ldots, q_r$ の何れかと一致する。ここで，r_k は p の倍数だから，$q_1, q_2, \ldots, q_k$ の何れかによって篩われる。これは r_t が篩われずに残っていることに矛盾する。故に，r_t は素数である。
r_2 は，$k = 0$ の場合で，残っている最小の数だから素数である。 □

さて，(篩表 n) で篩を実行しましょう。
篩は，$r_2 = \mathtt{Rp}_\mathtt{n}(2)$ から始めます。即ち，

[ステップ 1] $p = \mathtt{Rp}_\mathtt{n}(2)$ から篩い始める。

次に，順次大きな素数で篩っていきます。ある素数で篩ったとき，次に篩う素数は命題 5.1 より，篩った素数より大きい篩われていない最小の数です。それを p としましょう。即ち，

[ステップ 2] 次に篩う素数 p は篩った素数より大きい篩われていない最小の数。

更に，篩は (篩表 n) にある最大値 $\mathtt{Rp}_\mathtt{n}(\mathtt{MaxN})$ の平方根以下の素数で篩えば十分ですから，p は $\sqrt{\mathtt{Rp}_\mathtt{n}(\mathtt{MaxN})}$ 以下です。即ち，

[ステップ 3] 篩う素数 p は $p^2 \leqq \mathtt{Rp}_\mathtt{n}(\mathtt{MaxN})$ の範囲のもの。

次に，実際に p で篩を実行します。p で篩うとは，$m > 1$ となる整数に対して mp を表から取り除くことです。ここで，$m < p$ ならば，m の素因数 q は p より小さいので，既に篩に使った素数です。ですから，mp は q によって既に篩われた数です。ですから，この場合は篩う必要はありません。従って，$p \leqq m$ となる m について篩えば十分です。また，m が $\mathtt{mP}_\mathtt{n}$ と互いに素でなければ，mp は $\mathtt{mP}_\mathtt{n}$ と互いに素でなく，mp は $\mathtt{mP}_\mathtt{n}$ で篩われた数，即ち (篩表 n) にない数で，篩う必要はありません。
従って，m は $\mathtt{mP}_\mathtt{n}$ と互いに素，即ち，(篩表 n) にある数の場合だけ考えれば良いことになります。ですから，p の乗数としては，(篩表 n) にある p 以後の数を考えれば十分です。

[*50] ここで，$(m \mod 2)$ は m を 2 で割ったときの余りを示す。

ここで，$p = \mathrm{Rp_n}(i)$ と表されますから，p の乗数は (篩表 n) にある p 以後の数

$$\mathrm{Rp_n}(i),\ \mathrm{Rp_n}(i+1),\ \mathrm{Rp_n}(i+2),\ \mathrm{Rp_n}(i+3),\ \mathrm{Rp_n}(i+4),\ \ldots$$

で，篩う数は，

$$\mathrm{Rp_n}(i)\cdot p,\ \mathrm{Rp_n}(i+1)\cdot p,\ \mathrm{Rp_n}(i+2)\cdot p,\ \mathrm{Rp_n}(i+3)\cdot p,\ \mathrm{Rp_n}(i+4)\cdot p,\ \ldots$$

で十分にです。即ち，

[ステップ 4] 素数 $p = \mathrm{Rp_n}(i)$ に対して，

$$\mathrm{Rp_n}(i)\cdot p,\ \mathrm{Rp_n}(i+1)\cdot p,\ \mathrm{Rp_n}(i+2)\cdot p,\ \mathrm{Rp_n}(i+3)\cdot p,\ \mathrm{Rp_n}(i+4)\cdot p,\ \ldots$$

を篩えばよい。

ここで，(篩表 n) にある最大数は $\mathrm{Rp_n}(\mathtt{MaxN})$ ですから，その範囲で篩を実行すればよいことになります。即ち，

[ステップ 5] 素数 $p = \mathrm{Rp_n}(i)$ に対して，$j = 0, 1, 2, \ldots$

$$\mathrm{Rp_n}(i+j)\cdot p \leqq \mathrm{Rp_n}(\mathtt{MaxN})$$

の範囲で篩えばよい。

篩われた数を表に示すために配列変数 $\mathtt{NT}(i)$ の座標に戻すために，$\mathrm{Rp_n}(i+j)\cdot p$ に $\mathrm{Ir_n}(s)$ を施し，$\mathtt{NT}(i)$ に 1 を設定します。即ち，

[ステップ 6] 篩われた数であることを示すために，ステップ 5 の $i+j$ に対して，

$$\mathtt{NT}(\mathrm{Ir_n}(\mathrm{Rp_n}(i+j)\cdot p)) = 1$$

と設定する。

ステップ 1〜6 を纏めて，これらのことを擬似コードとして書いてみると，次のようになります。

```
'-----------------------------------
' 擬似コード 1
'-----------------------------------

i = 2
p = Rpn(i)
While p * p <= Rpn(MaxN)
      i0 = i
      While Rpn(i0) * p <= MaxN
            NT(Irn(Rpn(i0) * p)) = 1
            i0 = i0 + 1
      End While
      i = i + 1
      While NT(i) = 1 : i = i + 1 : End While
      p = Rpn(i)
End While

'-----------------------------------
```

ステップ 5 での j の上限値 j_0 を具体的に求めましょう。上限値 j_0 は，定義から，

$$\mathrm{Rp_n}(i+j_0)\cdot p \ \leqq\ \mathrm{Rp_n}(\mathrm{MaxN}) < \mathrm{Rp_n}(i+j_0+1)\cdot p$$

となる j_0 です。これを p で割ると，

$$\mathrm{Rp_n}(i+j_0) \ \leqq\ \frac{\mathrm{Rp_n}(\mathrm{MaxN})}{p} < \mathrm{Rp_n}(i+j_0+1)$$

となります。ここで，$\mathrm{Rp_n}(i+j_0)$，$\mathrm{Rp_n}(i+j_0)$ は整数ですから，中央式にガウス記号を施してもこの不等式は成立します。即ち，

$$\mathrm{Rp_n}(i+j_0) \ \leqq\ \left[\frac{\mathrm{Rp_n}(\mathrm{MaxN})}{p}\right] < \mathrm{Rp_n}(i+j_0+1)$$

となります。更に，$\mathrm{Floor_n}(m)$ は m 以下で $\mathrm{Rp_n}(i)$ の形の最大数ですから，

$$\mathrm{Rp_n}(i+j_0) \ \leqq\ \mathrm{Floor_n}\left(\left[\frac{\mathrm{Rp_n}(\mathrm{MaxN})}{p}\right]\right) \leqq \left[\frac{\mathrm{Rp_n}(\mathrm{MaxN})}{p}\right] < \mathrm{Rp_n}(i+j_0+1)$$

より，

$$\mathrm{Rp_n}(i+j_0) \ =\ \mathrm{Floor_n}\left(\left[\frac{\mathrm{Rp_n}(\mathrm{MaxN})}{p}\right]\right)$$

となります。この両辺に $\mathrm{Rp_n}$ の逆関数 $\mathrm{Ir_n}$ を施すと，

$$i+j_0 = \mathrm{Ir_n}\left(\mathrm{Floor_n}\left(\left[\frac{\mathrm{Rp_n}(\mathrm{MaxN})}{p}\right]\right)\right)$$

となります。この上限値を使って，上の擬似コードを書き直すと，次のようになります。

```
'-----------------------------------
' 擬似コード 2
'-----------------------------------

i = 2
p = Rpn(i)
While p * p <= Rpn(MaxN)
      iMax = Irn(Floorn(Int(MaxN/p)))
      For i0 = i To iMax
            NT(Irn(Rpn(i0) * p)) = 1
      Next i0
      i = i + 1
      While NT(i) = 1 : i = i + 1 : End While
      p = Rpn(i)
End While

'-----------------------------------
```

5.4 $n = 2,\ \mathtt{mP}_2 = 2\cdot 3 = 6$ の場合の $\mathtt{Rp}_n, \mathtt{Ir}_n, \mathtt{Floor}_n$ の計算

前項では，$\mathtt{Rp}_n, \mathtt{Ir}_n, \mathtt{Floor}_n$ を利用して篩の擬似コードを比較的簡単に書くことができました。

また関数 $\mathtt{Rp}_n, \mathtt{Ir}_n, \mathtt{Floor}_n$ の具体的な形は，$n = 0, 1,\ \mathtt{mP}_n = 1, 2$ の場合は例 5.2 より簡単に求められました。そこで，次に，$n = 2,\ \mathtt{mP}_n = 2\cdot 3 = 6$ の場合を考えてみましょう。

(表2) にある数列は，$2, 3$ と互いに素ですから，$6k+1,\ 6k+5 = 6(k+1)-1$ の形の数であることが分かります。更に，これらは $6k$ を中心に ± 1 の対が並ぶことから，整数 i を 2 進数とみて，1 桁目が，± 1 を示し，2 桁以上の数が $6k$ を表すとみると，(表2) の数列は式で表すことができます。

これらのことを纏めると，$\mathtt{rp}_2, \mathtt{ir}_2$ は具体的に次で与えられることが分かります。

命題 5.2. $\mathtt{mP}_2$，即ち，(表 2) の場合，

(1) $i = 2, 3, \ldots$ に対して，

$$\mathtt{Rp}_2(i) = 6\left[\frac{i}{2}\right] - (-1)^i \tag{5.2.1}$$

で表される。

(2) 逆に，このようにして得られる数列 $r = 5, 7, \ldots$ に対して，その逆 i は

$$\mathtt{Ir}_2(r) = 2\left[\frac{r-1}{6}\right] + (r \bmod 3) \tag{5.2.2}$$

で与えられる。

(3) $m = 1, 2, 3, \ldots$ に対して，

$$\mathtt{Floor}_2(m) \quad = \quad 6\left[\frac{m-1}{6}\right] + 4\left[\frac{((m+5) \bmod 6)+1}{5}\right] + 1, \tag{5.2.3}$$

となる。

(4) $m = 1, 2, 3, \ldots$ に対して，

$$\mathtt{Ceil}_2(m) \quad = \quad 6\left[\frac{m}{6}\right] + 4\left[\frac{(m \bmod 6)+2}{4}\right] + 1, \tag{5.2.4}$$

となる。

証明.

(1) まず，$2, 3$ と互いに素な整数列 $j = 5, 7, \ldots$ は $6k+1,\ 6k+5 = 6(k+1)-1$ の形である。
実際，すべての整数は $6k + r\ (r = 0, 1, 2, 3, 4, 5)$ の形に表される。$r = 0, 2, 3, 4$ の場合はそれぞれ，$6, 2, 3, 2$ で割り切れ，2 または 3 と素でない。また $r = 1, 5$ の場合は，$6k+1,\ 6k+5$ の形で，$2, 3$ と互いに素である。
式 (5.2.1) の右辺はこの形の数だから，$2, 3$ と互いに素な数列を表す。即ち，(1.1) の右辺は $i = 2, 3, \ldots$ から (表2) への写像を定める。そこで，この写像が狭義単調増加数列であること，および，全射であることを示す[*51]。これにより，(5.2.1) の右辺が $\mathtt{Rp}_2$ の定義に条件を満たすことが分かり，(5.2.1) が成立することが示される。

[*51] 狭義の単調増加関数は単射だから，これで，(5.2.1) の右辺が全単射であることが示される。

(i) **単調増加性。**i を 2 で割った余りを s ($s = 0$または1) とすると，$i = 2k + s$ と表される。このとき，(1) は，

$$j_i = 6k - (-1)^s$$

と書くことができる。$i < i'$ となる i' をとり，同様に，$i' = 2k' + s'$ と表す。このとき，$k < k'$ または，$s < s'$ となる。$k < k'$ ならば，$6k - (-1)^r < 6k' - (-1)^{k'}$ であり，$j_i < j_{i'}$ となる。更に，$k = k'$, $s < s'$ ならば，$s = 0$, $s' = 1$ だから，$j_i = 6k - (-1)^s = 6k' - 1 < 6k' + 1 = 6k' - (-1)^1 = j_{i'}$ となる。

(ii) **全射性。**$j = 6k + 1$ の場合，$i = 2k + 1$ とすると，

$$j_i = 6\left[\frac{2k+1}{2}\right] - (-1)^{(2k+1)} = 6k + 1 = j$$

となる。$j = 6k + 5 = 6(k + 1) - 1$ の場合，$i = 2(k + 1)$ とすると，

$$j_i = 6\left[\frac{2(k+1)}{2}\right] - (-1)^{(2(k+1))} = 6(k + 1) - 1 = 6k + 5 = j$$

となる。

(2) 次に，式 (5.2.2) の右辺式 (5.2.1) の逆を与えることを示す。

(i) i が奇数，即ち，$i = 2k + 1$ の場合，(5.2.1) より，

$$j_i = 6\left[\frac{i}{2}\right] - (-1)^i = 6\left[\frac{2k}{2}\right] - (-1)^{2k+1} = 6k + 1 = j$$

となる。このとき，(5.2.2) より，

$$i_j = 2\left[\frac{j-1}{6}\right] + (j \bmod 3) = 2k + 1 = i$$

となる。

(ii) i が偶数，即ち，$i = 2k$ の場合，(5.2.1) より，

$$j_i = 6\left[\frac{i}{2}\right] - (-1)^i = 6\left[\frac{2k}{2}\right] - (-1)^{2k} = 6k - 1 = 6(k - 1) + 5 = j$$

となる。このとき，(5.2.2) より，

$$i_j = 2\left[\frac{j-1}{6}\right] + (j \bmod 3) = 2(k - 1) + 2 = 2k = i$$

となる。

以上から，(5.2.2) の右辺が，(表2) 上で $\mathtt{Rp}_2$ の逆を表すこと分かる。さらに，$\mathtt{Rp}_2$ は全単射で逆写像 $\mathtt{Ir}_n$ の存在は既知なので，(5.2.2) の右辺が $\mathtt{Ir}_2$ を表すことが示された。

(3) $m = 1, 2, 3, \ldots$ に対して，$m = 6k + r$ $(0 \le r < 6)$ と表すと，定義から，

	r	0	1	2	3	4	5
	$\mathtt{Floor}_2(m)$	$m-1$	m	$m-1$	$m-2$	$m-3$	m
$(*0)$	$=$	$6(k-1)+5$	$6k+1$	$6k+1$	$6k+1$	$6k+1$	$6k+5$

となる。

一方，r のそれぞれの場合について，(5.2.3) の式に現れる各項を計算してみると，

	r	0	1	2	3	4	5
	$m-1$	$6(k-1)+5$	$6k$	$6k+1$	$6k+2$	$6k+3$	$6k+4$
	$m+5$	$6k+5$	$6(k+1)$	$6(k+1)+1$	$6(k+1)+2$	$6(k+1)+3$	$6(k+1)+4$
$(*1)$	$\left\lfloor \frac{m-1}{6} \right\rfloor$	$k-1$	k	k	k	k	k
$(*2)$	$((m+5) \bmod 6)+1$	6	1	2	3	4	5
$(*3)$	$\left\lfloor \frac{(*2)}{5} \right\rfloor$	1	0	0	0	0	1
$(*4)$	$6(*1)+4(*3)+1$	$6(k-1)+5$	$6k+1$	$6k+1$	$6k+1$	$6k+1$	$6k+5$

となる。ここで，$(*4) = (*0)$ であるから，(5.2.3) が成立することが分かる。

(4) $m = 1, 2, 3, \ldots$ に対して，$m = 6k + r$ $(0 \le r < 6)$ と表すと，定義から，

	r	0	1	2	3	4	5
	$\mathtt{Ceil}_2(m)$	$m+1$	m	$m+3$	$m+2$	$m+1$	m
$(*0)$	$=$	$6k+1$	$6k+1$	$6k+5$	$6k+5$	$6k+5$	$6k+5$

となる。

一方，r のそれぞれの場合について，(5.2.3) の式に現れる各項を計算してみると，

	r	0	1	2	3	4	5
	m	$6k$	$6k+1$	$6k+2$	$6k+3$	$6k+4$	$6k+5$
$(*1)$	$\left\lfloor \frac{m}{6} \right\rfloor$	k	k	k	k	k	k
$(*2)$	$(m \bmod 6)+2$	2	3	4	5	6	7
$(*3)$	$\left\lfloor \frac{(*2)}{4} \right\rfloor$	0	0	1	1	1	1
$(*4)$	$6(*1)+4(*3)+1$	$6k+1$	$6k+1$	$6k+5$	$6k+5$	$6k+5$	$6k+5$

となる。ここで，$(*4) = (*0)$ であるから，(5.2.4) が成立することが分かる。

□

命題 5.2 を使えば擬似コード 2 から 10000 以下の素数表を作るプログラムは以下の通り，簡単に書くことができます。ここで，篩う上限を MaxSN と表すと，MaxN = Ir_n(FloorN(MaxSN)) となります。ですから，上の擬似コードにあった Rp_n(MaxN) は，FloorN(MaxSN) と書くこともできます。篩う上限を最初に決めることからすれば，こちらの方が分かり易いでしょう。また，実際には MaxSN でも構いません。

[*52]

プログラム **5.2**

```
'-----------------------------------
' 既約篩法：mP2=2*3=6 の場合,
' 10000 以下の素数表の作成
'-----------------------------------
Const NP = 2
Const MaxSN = 10000
Const MaxN = 3333   :' Ir2(Floor2(MaxSN))
Dim NT(MaxN)
i=2
p=Rp2(i)
sMaxN=Floor2(MaxSN) :' sMaxN=MaxSN でも OK
While p*p<= sMaxN
   iMax = Ir2(Floor2(Int(sMaxN/p)))
   For i0=i to iMax
      NT(Ir2(Rp2(i0)*p))=1
   Next i0
   i=i+1
   While NT(i)=1:i=i+1:Wend
   p=Rp2(i)
Wend
Counter = NP
Print 2;" ";3;" ";
For i=2 to MaxN
   If NT(i)=0  then
      Counter= Counter +1
      If (Counter mod 10) = 0 then
         Print rp2(i)
      Else
         Print rp2(i); " ";
      End if
   End If
Next i

End

Function Rp2(i)
   Rp2 = 6*Int(i/2) - (-1)^(i mod 2)
End Function

Function Ir2(j)
   Ir2 = 2*Int((j-1)/6) + (j mod 3)
End Function

Function Floor2(i)
   Floor2 = 6*Int((i-1)/6) + 4*Int((((i + 5) mod 6) + 1)/5) + 1
End Function
'-----------------------------------
```

[*52] この方法は mP_2 と互いに素な数に篩を施します・そして，これらの数は mP_2 を法にして，既約剰余類を構成することから，この方法を既約篩法 (reduced Sieve) と言うことにします。

5.5 表による $\mathtt{Rp}_n, \mathtt{Ir}_n, \mathtt{Floor}_n, \mathtt{Ceil}_n$ の計算

前項では，$n=2,\ \mathtt{mP}_2 = 2\cdot 3 = 6$ の場合の $\mathtt{Rp}_n, \mathtt{Ir}_n, \mathtt{Floor}_n, \mathtt{Ceil}_n$ を表す式を求めました。

これらの式は $\mathtt{mP}_n$ またはその約数を法として決定されることを注意すれば，更に大きな $\mathtt{mP}_n$ についても類似な式を求めることができるでしょう。しかし，それらは，あまり簡単な式とは言えず，その値の計算も余り効率的とは言えないでしょう。

篩は素数を，纏めて，効率的，高速に計算できることにその存在意義があります。時間がかかってもよいのであれば，素朴に小さい数から割っていって素数を決定する方が簡単でしょう。ですから，$\mathtt{Rp}_n, \mathtt{Ir}_n, \mathtt{Floor}_n, \mathtt{Ceil}_n$ も効率的に計算できることが重要になります。

命題 5.2 の証明では，関数式が成立することを $\pmod 6$ での場合分けの表を利用しました。そこで関数を式として表現し，それから値を計算するのではなく，表から関数値を計算することを考えてみましょう。

まず，$n=2,\ \mathtt{mP}_2 = 2\cdot 3 = 6$ の場合を小さい値について実際に表にして見ると，次が得られます。

i	1	2	3	4	5	6	7	8
$\mathtt{Rp}_2(i)$	1	5	7	11	13	17	19	23
$6\left[\frac{i}{2}\right]$	0	6	6	12	12	18	18	24
$\mathtt{i} \bmod 2$	1	0	1	0	1	0	1	0

r	1	5	7	11	13	17	19	23
$\mathtt{Ir}_2(r)$	1	2	3	4	5	6	7	8
$2\left[\frac{r}{6}\right]$	0	0	2	2	4	4	6	6
$\mathtt{r} \bmod 6$	1	5	1	5	1	5	1	5

この結果から，表を使って $\mathtt{rp}_2(i),\ \mathtt{ir}_2(r)$ が次の式で計算できることが分かります。

例 5.3. $n=2,\ \mathtt{mP}_2 = 2\cdot 3 = 6$ の場合 $\mathtt{rP}(0) = -1,\ \mathtt{rP}(1) = 1$ とすると，$i = 1, 2, \ldots$ に対して

$$\mathtt{Rp}_2(i) = 6\left[\frac{i}{2}\right] + \mathtt{rP}(i \bmod 2) \tag{5.2.1'}$$

で与えられる。更に，$\mathtt{iR}(1) = 1,\ \mathtt{iR}(5) = 2$ とすると，$r = 1, 5, 7, 11 \ldots$ に対して

$$\mathtt{Ir}_2(r) = 2\left[\frac{r}{6}\right] + \mathtt{iR}(r \bmod 6) \tag{5.2.2'}$$

で与えられる。

証明. (1) (5.2.1′)。$\mathtt{rP}(i \bmod 2) = -(-1)^i$ であり，(5.2.1′) 右辺第 2 項と (5.2.1) の右辺第 2 項の値が一致することから確かめられる。

(2) (5.2.2′)。$r = 6k+1,\ 6k+5$ に対して，(5.2.2′) 右辺第 2 項と (5.2.2) 右辺第 2 項がそれぞれ，$2k+1,\ 2(k+1)$ と値が一致することから分かる。

□

一般に次が成立します。

命題 5.3. $\mathtt{mP_n} = p_1 p_2 \cdots p_n$ とし，$\mathtt{phiP_n} = \phi(\mathtt{mP_n})$ と置く。

(1) $r_1 = 1 < r_2 < \ldots < r_{\mathtt{phiP_n}}$ を $\mathtt{mP_n}$ 以下の $\mathtt{mP_n}$ と互いに素な正整数とする。配列変数 $\mathtt{rP}$ に対して，$\mathtt{rP}(0) = -1,\ \mathtt{rP}(i) = r_i\ (i = 1, 2, 3, \ldots, \mathtt{phiP_n})$ と設定する。このとき，

$$\mathtt{Rp_n}(i) = \mathtt{mP_n}\left[\frac{i}{\mathtt{phiP_n}}\right] + \mathtt{rP}(i \bmod \mathtt{phiP_n}) \tag{5.3.1}$$

が成立する[*53]。

(2) 配列変数 $\mathtt{iR}(0)$〜$\mathtt{iR}(\mathtt{mP_n} - 1)$ を用意し，$i = 1, 2, \ldots, \mathtt{phiP_n}$ に対して，$\mathtt{iR}(\mathtt{rP}(i)) = i$ とする[*54]。このとき，

$$\mathtt{Ir_n}(r) = \mathtt{phiP_n}\left[\frac{r}{\mathtt{mP_n}}\right] + \mathtt{iR}(r \bmod \mathtt{mP_n}) \tag{5.3.2}$$

が成立する。

(3) 配列変数 $\mathtt{fL}(0)$〜$\mathtt{fL}(\mathtt{mP_n} - 1)$ を用意し，$\mathtt{fL}(0) = -1,\ \mathtt{fL}(i) = \mathtt{Floor_n}(i) - i\ (i = 1, 2, \ldots, \mathtt{mP_n} - 1)$ と設定すると，$i = 1, 2, \ldots$ に対して，

$$\mathtt{Floor_n}(i) = i + \mathtt{fL}(i \bmod \mathtt{mP_n}) \tag{5.3.3}$$

が成立する。

(4) 配列変数 $\mathtt{cE}(0)$〜$\mathtt{cE}(\mathtt{mP_n} - 1)$ を用意し，$\mathtt{cE}(0) = 1,\ \mathtt{cE}(i) = \mathtt{Ceil_n}(i) - i\ (i = 1, 2, \ldots, \mathtt{mP_n} - 1)$ と設定すると，$i = 1, 2, \ldots$ に対して，

$$\mathtt{Ceil_n}(i) = i + \mathtt{cE}(i \bmod \mathtt{mP_n}) \tag{5.3.4}$$

が成立する。

証明. (1) まず，$i = \mathtt{phiP_n}k + t\ (0 \le t < \mathtt{phiP_n})$ と表すと，(5.3.1) の右辺は $\mathtt{mP_n}k + \mathtt{rP}(t)$ と表される。そこで，(5.3.1) の右辺が $\mathtt{rp_n}$ の定義を満たすことを示す。$\mathtt{rP}(i)$ の定め方から，$\mathtt{rP}(i)$ は $\mathtt{mP_n}$ と互いに素である。従って，(5.3.1) の右辺は (表n) の元を表す。逆に，r を (表n) の元とし，$r = \mathtt{mP_n}k + s\ (0 \le s < \mathtt{mP_n})$ の形に表すと，s は $\mathtt{mP_n}$ と互いに素だから，$s = \mathtt{rP}(j)$ の形に表され，(5.3.1) の右辺が $i = 1, 2, \ldots$ から (表n) への全単射を定めることが分かる。

更に，$\mathtt{rP}(i)$ は単調増加で，$0 < \mathtt{rP}(i) < \mathtt{mP_n}$ だから，(5.3.1) の右辺は単調増加関数である。従って，これは $\mathtt{rp_n}$ に他ならない。

(2) (1) の証明と同様に，$i = \mathtt{phiP_n}k + t\ (0 \le t < \mathtt{phiP_n})$ と表すと，(1) の結果より，$r = \mathtt{Rp_n}(i) = \mathtt{mP_n}k + \mathtt{rP}(t)$ となる。この r を (5.3.2) の右辺に代入すると，$\mathtt{phiP_n}k + t = i$ が得られる。従って，(5.3.2) の右辺は $\mathtt{Rp_n}$ の逆を与える。故に，(5.3.2) が成立する。

(3) $i = k\mathtt{mP_n} + s\ (0 \leqq s < \mathtt{mP_n})$ の形に表すと，(5.3.3) の右辺は $\mathtt{fL}$ の定め方から，$k\mathtt{mP_n} + s + \mathtt{Floor_n}(s) - s = k\mathtt{mP_n} + \mathtt{Floor_n}(s)$ となる。ここで，$\mathtt{Floor_n}(s)\ (\leqq s)$ は $\mathtt{mP_n}$ と互いに素だから，(5.3.3) の右辺は $\mathtt{mP_n}$ と互いに素になる。更にもし，$k\mathtt{mP_n} + \mathtt{Floor_n}(s) < t \leqq k\mathtt{mP_n} + s$ となる $\mathtt{mP_n}$ と互いに素な t が存在したとすると，$k\mathtt{mP_n} + t'\ (0 \leqq t' < \mathtt{mP_n})$ と表され，$\mathtt{Floor_n}(s) < t' \leqq s$ かつ，t' は $\mathtt{mP_n}$ と互いに素になる。これは $\mathtt{Floor_n}(s)$ の定義に反する。従って，$k\mathtt{mP_n} + \mathtt{Floor_n}(s)$ は $i = k\mathtt{mP_n} + s$ 以下最大の $\mathtt{mP_n}$ と互いに素な数，即ち $\mathtt{Floor_n}(i)$ である。故に (5.3.3) が成立する。

[*53] 上の式では，$\mathtt{rP}(\mathtt{phiP_n})$ は使用していませんが，以下で $\mathtt{Ir_n}$ の計算に使います。

[*54] これ以外即ち，$\mathtt{mP_n}$ と互いに素でない s に対しては，$\mathtt{iR}(s) = 0$ とする

(4) (3) と同様に証明される。$i = k\mathtt{mP_n} + s$ $(0 \leqq s < \mathtt{mP_n})$ の形に表すと，(5.3.4) の右辺は cE の定め方から，$k\mathtt{mP_n} + s + \mathtt{Ceil_n}(s) - s = k\mathtt{mP_n} + \mathtt{Ceil_n}(s)$ となる。ここで，$\mathtt{Ceil_n}(s)$ $(\geqq s)$ は $\mathtt{mP_n}$ と互いに素だから，(5.3.4) の右辺は $\mathtt{mP_n}$ と互いに素になる。更にもし，$k\mathtt{mP_n} + s \leqq t < k\mathtt{mP_n} + \mathtt{Ceil_n}(s)$ となる $\mathtt{mP_n}$ と互いに素な t が存在したとすると，$k\mathtt{mP_n} + t'$ $(0 \leqq t' < \mathtt{mP_n})$ と表され，$s \leqq t' < \mathtt{Ceil_n}(s)$ かつ，t' は $\mathtt{mP_n}$ と互いに素になる。これは $\mathtt{Ceil_n}(s)$ の定義に反する。従って，$k\mathtt{mP_n} + \mathtt{Ceil_n}(s)$ は $i = k\mathtt{mP_n} + s$ 以上最小な $\mathtt{mP_n}$ と互いに素な数，即ち $\mathtt{Ceil_n}(i)$ である。故に (5.3.4) が成立する。

□

上の命題で使われた，配列変数 rP, iR, fL, cE を具体的な場合で求めてみると，次のようになります。

例 **5.4.** (1) $\mathtt{mP_2} = 2 \cdot 3 = 6$ の場合。このとき，$\mathtt{phiP_2} = 2$ で，rP は rP(0)～rP(2)，iR, fL, cE はそれぞれ 0～5 の配列変数で，値はそれぞれ次に設定する

i	0	1	2
rP(i)	−1	1	5

i	0	1	2	3	4	5
rR(i)	0	1	0	0	0	2

i	0	1	2	3	4	5
fL(i)	−1	0	−1	−2	−3	0

i	0	1	2	3	4	5
cE(i)	1	0	3	2	1	0

(2) $\mathtt{mP_3} = 2 \cdot 3 \cdot 5 = 30$ の場合。このとき，$\mathtt{phiP_3} = 8$ で，rP は rP(0)～rP(8)，iR, fL, cE はそれぞれ 0～29 の配列変数で，値はそれぞれ次に設定する[*55]。

i	0	1	2	3	4	5	6	7	8
rP(i)	−1	1	7	11	13	17	19	23	29

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
iR(i)	0	1	0	0	0	0	0	2	0	0	0	3	0	4	0
i	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
iR(i)	0	0	5	0	6	0	0	0	7	0	0	0	0	0	8

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
fL(i)	−1	0	−1	−2	−3	−4	−5	0	−1	−2	−3	0	−1	0	−1
i	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
fL(i)	−2	−3	0	−1	0	−1	−2	−3	0	−1	−2	−3	−4	−5	0

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
cE(i)	1	0	5	4	3	2	1	0	3	2	1	0	1	0	3
i	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
cE(i)	2	1	0	1	0	3	2	1	0	5	4	3	2	1	0

[*55] $\mathtt{rP}(8) = 29$ であるが，$8 \pmod{\mathtt{phiP_3}} = \mathtt{0}$ より，$\mathtt{rP}(0) = -1 \equiv 29 \pmod{\mathtt{mP_3}}$ と設定する。

上の例の場合，実際にそれらのデータをプログラムに書き込んで利用することができます。しかし，それより mP_n が大きい場合，例えば，$mP_5 = 2\cdot3\cdot5\cdot7\cdot11 = 2310$ のような場合，データが多いので，表を実際にデータとして保存するのは余り効率的ではありません。rP や iR の配列変数は必要ですが，実は，それらの保持する値は，簡単に計算できます。

例えば，rP，iR は次のプログラムで計算できます。

```
'-----------------------------------
' rP，iR 作成擬似プログラム
'-----------------------------------

Counter = 0
rP(0) = -1
For i = 1 to mPn
      If GCD(mPn, i) = 1 then
            Counter = Counter + 1
            rP(Counter) = i
      End If
Next i

'-----------------------------------

For i = 1 to mPn
      iR(rP(i)) = i
Next i

'-----------------------------------
```

また，fL，cE も例えば次のプログラムで作成できます。

```
'-----------------------------------
' fL，cE 作成擬似プログラム
'-----------------------------------

For i = 1 to mPn - 1
      If Gcd(i, mPn) = 1 then
            fD = 0
      Else
            fD = fD - 1
      End If
      fL(i) = fD
Nexti

'-----------------------------------

For i = 1 to mPn - 1
      If Gcd(mPn - i, mPn) = 1 then
            cD = 0
      Else
            cD = cD + 1
      End If
      cE(mPn - i) = cD
Nexti

'-----------------------------------
```

このように，プログラムはいずれも簡単ですから，mP_n が 30 を超える場合は，rP，iR，fL，cE の値はプログラムで計算させるのがよいでしょう。

5.6 既約篩法：$mP_3 = 2 \cdot 3 \cdot 5 = 30$ の場合のプログラム例

前節の結果を利用して実際にプログラムを書いてみましょう。次のプログラムは mP_3 と互いに素な表 (表3) を使って[*56]，10^7 以下の素数表を求めるものです。

定数設定では，mP_n での n を NP と表しています。mP_3 を使うので，NP = 3 とします。mPn，phiPn はそれぞれ，mP_3，$phiP_3$ を表します。これらを使って，配列変数 rP(phiPn)，iR(mPn)，fL(mPn) をそれぞれ宣言します。それらの具体的値は，今回はデータ文でプログラムに直接記述し，それを Read 文で読み込んで使います。

MaxSN = 10^7 ですが，MaxN は，10^7 までの素数表を作ることから，$Ir_n(Floor_n(10^7))$ の値を事前に計算し，それをプログラムに書き込みます[*57]。

篩の部分は（擬似コード 2）ほぼそのままです。関数 Rp_n，Ir_n，$Floor_n$ は命題 5.3 を使って表示しています。

結果表示の部分は，素数の数が多いので，実際には素数を表示していません。ここでは素数の個数のみを計算して表示しています。

プログラム **5.3**

```
'-----------------------------------------------
'   9999997(10000000=10^7) 以下の素数表の作成
'-----------------------------------------------
' 既約篩法 2,3,5 と互いに素な表を使って篩う
Public Const  NP = 3
Public Const  mPn = 30 : ' 30=2*3*5
Public Const  phiPn = 8  : ' phi(30)=8

Public rP(phiPn)
Public iR(mPn)
Public fL(mPn)

' rP(i) 用データ
Data -1,1,7,11,13,17,19,23,29
For i=0 To phiPn: Read rP(i): Next i

' iR(i) 用データ
Data 0, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 3, 0, 4, 0
Data 0, 0, 5, 0, 6, 0, 0, 0, 7, 0, 0, 0, 0, 0, 8
For i=0 To mPn-1: Read iR(i): Next i

' fL(i) 用データ
Data -1, 0,-1,-2,-3,-4,-5, 0,-1,-2,-3, 0,-1, 0,-1
Data -2,-3, 0,-1, 0,-1,-2,-3, 0,-1,-2,-3,-4,-5, 0
For i=0 To mPn-1: Read fL(i): Next i

Const MaxSN=10000000 :' 10^7
Const MaxN=2666666: ' Irn(FloorN(10^7))

Dim NT(MaxN)
Print Time$
i=2
p=Rpn(i)
```

[*56] (表3) は本文にはありませんが，(表n) の $n = 3$ の場合を意図しています。

[*57] この値は MaxN に適当な値，例えば，1000 を設定してプログラムを動かし，ダイレクトモードで Print irn(FloorN(10^7)) と入力して求めることができます。

```
sMaxN=MaxSN
While p*p<= sMaxN
'   Print p;"-Sieving"
   iMax = Irn(FloorN(Int(sMaxN/p)))
   For i0=i To iMax
      NT(Irn(Rpn(i0)*p))=1
   Next i0
   i=i+1
   While NT(i)=1:i=i+1:End While
   p=Rpn(i)
End While
Print Time$
Counter = NP
For i=2 To MaxN
'   必要なら 素数を表示の処理を記述
   If NT(i)=0  then Counter= Counter +1
Next i
Print "the Number of Primes less than"; Rpn(MaxN);" is ";Counter;"."
End

Function Rpn(i)
   Rpn = mPn * Int(i/phiPn) + rP(i mod phiPn)
End Function

Function Irn(j)
   Irn = phiPn * Int(j/mPn) + iR(j mod mPn)
End Function

Function FloorN(i)
   FloorN= i + fL(i mod mPn)
End function
'---------------------------------
```

5.7 既約篩法 $\mathrm{mP}_7 = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 = 510510$ の場合のプログラム例

$\mathrm{mP}_7 = 510510$ と比較的大きい場合の既約篩法の例プログラムを挙げましょう。この場合は，rP, iR, fL 等はデータとしてプログラムに記述することは不向きです。そこでここでは前述の rP, iR, fL 作成擬似プログラムほぼそのまま使って計算しています。

また，MaxSN はここでは，10^6 で比較的小さいので，素数表を画面表示させています。

プログラム **5.4**

```
'---------------------------------
' 既約篩法 n=7：10^6 までの素数表の作成
'---------------------------------
' 2,3,5,7,11,13,17(510510) と互いに素な表を使って篩う
Public Const mPn=510510
Public Const phiPn=92160
' 1*2*4*6*10*12*16 = phi(510510)
Public Const NP = 7
Public rP(phiPn)
Public iR(mPn)
Public fL(mPn)

Const MaxSN = 1000000:' 10000000=10^6
Const MaxN = 180524:'  Irn(FloorN(10^6))
```

```
Dim NT(MaxN)
Call MakeArrayData
Print Time$
i=2
p=Rpn(2)
sMaxN=MaxSN
While p*p<= sMaxN
   iMax = Irn(FloorN(Int(sMaxN/p)))
   For i0=i To iMax
      NT(Irn(Rpn(i0)*p))=1
   Next i0
   i=i+1
   While NT(i)=1:i=i+1:End While
   p=Rpn(i)
End While
Print Time$
Counter = NP
For i=2 To MaxN
   If NT(i)=0  then
      Counter= Counter +1
      If (Counter mod 10) = 0 then
         Print Rpn(i)
      Else
         Print Rpn(i); " ";
      End If
   End If
Next i
Print
Print "the Number of Primes less than"; Rpn(MaxN);" is ";Counter;"."
End

Function Rpn(i)
   Rpn = mPn* Int(i/phiPn) + rP(i mod phiPn)
End Function

Function Irn(j)
   Irn = phiPn*Int(j/mPn) + iR(j mod mPn)
End Function

Function FloorN(i)
   FloorN= i +fL(i mod mPn)
End function

Sub MakeArrayData
Print Time$;
Print "' rP Data の作成"
   Counter = 0
   rP(0)=-1
   For i=1 To mPn
      If Gcd(mPn,i)=1 then
         Counter = Counter + 1
          rP(Counter)=i
      End If
   Next i
Print Time$;
Print "' iR Data の作成"
   For i=1 To phiPn
      iR(rP(i))=i
   Next i
```

```
Print Time$;
Print "' fL Data の作成"
   fL(0)=-1
   For i=1 To mPn-1
      If Gcd(i,mPn)=1 then
         fD=0
      Else
         fD=fD -1
      End If
      fL(i)=fD
   Next i
End Sub
'-----------------------------------
```

5.8 どの篩法，既約篩法ではどの mP_n を使うべきか

今までいくつかのエラトステネス系の篩を説明してきました。それらを列挙すると次のようになります。

- プログラム 2.1'：篩法 1 による最も基本的な方法
- プログラム 2.2'：篩法 1 の改良版
- プログラム 3.1：車輪篩法 (w_4)
- プログラム 4.1：リスト篩法
- プログラム 5.1'：奇数篩法
- プログラム 5.2：既約篩法 ($mP_2 = 6$)：Rp_n, Ir_n の式利用版
- プログラム 5.3：既約篩法 ($mP_3 = 30$)：表計算版（表 Data の事前計算)
- プログラム 5.4：既約篩法 ($mP_7 = 510510$)：表計算版

ここではそれらの篩法を比較して，それらの特徴を考えてみましょう。比較する規準としては次を考えます。

- プログラムの簡単さ，単純さ
- 実行速度
- 篩える最大数

■プログラムの簡単さ，単純さ

プログラムの簡単さ，単純さを測る明確な規準はありませんが，今の場合，プログラムの長さを比べるとある程度分かります[*58]。

そこで，上のプログラムの行数を本質的部分の行数を数えてみます。具体的には，プログラムのコメント行，空行，時間，結果の表示用の行は数えないことにします[*59]。表にしてみると，次のようになります。

プログラム	2.1′	2.2′	3.1(w_4)	4.1	5.1′	5.2($n=2$)	5.3($n=3$)	5.4($n=7$)
行数	10行	13行	36行	20行	10行	26行	38行	50行

プログラムの長さ，複雑さを比較すれば，プログラム 2.1'，5.1'，2.2'，4.1，5.2，3.1，5.3，5.4 の順になります。

[*58] 勿論一般的に，プログラムの長さでプログラムの簡単さ，単純さを測ることはできません。しかし今回の場合はプログラムの目標，プログラム技法は同じで，違うのは使うアルゴリズムの違いですから，それを実現するプログラムの行数がアルゴリズムの複雑さを示す一つの尺度と考えられます。

[*59] 数え方により，プログラムの書き方により多少の違いはありますので，以下の数字は概数と考えてください。

■実行速度

次に，それぞれの実行速度を比較してみましょう。ここではプログラムを配列宣言部分を書き換えて 10^8 までの素数表を作成するものにします。

プログラム 5.4 は表を使って $\mathtt{Rp_n}, \mathtt{Ir_n}, \mathtt{Floor_n}$ の計算を行う既約篩法のプログラムですが，使用する表もプログラムの中で計算をしていて，宣言部分を少し書き直すだけで，$n = 1, 2, 3, \ldots, 8$ に対するプログラムとすることができます[*60]。

それらの実行時間を私の環境で計った結果は以下の通りです。

[10^8 までの素数表作成]

プログラム	2.1′	2.2′	3.1($\mathtt{w_4}$)	4.1	5.1′	5.2($n=2$)	5.3($n=3$)	5.4($n=7$)
時間	2分	1分32秒	48秒	(2分40秒)	2分24秒	1分31秒	57秒	30秒

[*61]

速度的には車輪篩法は既約篩法とほぼ同等な効率が得られます[*62]。車輪篩法は $\mathtt{w_k}$ の k を大きくすると速度が上がりましたが，既約篩法も同様なことが言えます。以下はプログラム 5.4 をもとにした同じく，10^8 までの素数表作成の既約篩法での n による比較です。

既約篩法	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$
時間	2分28秒	1分22秒	58秒	45秒	37秒	34秒	30秒	27秒

但し，$n = 8$ の場合，表の作成に約 14 秒かかります。$n = 7$ 以下の場合，表は 1 秒未満で作成できます。

■篩える最大数

大きな素数表を作るための篩では，それに対応した大きな配列を使います。例えば，プログラム 2.1 で n までの素数表を作るには，n 個の配列変数が必要です。

どのようなプログラミング言語を使っても，使用できる配列変数の個数には限りがあります。ですから大きな配列変数を扱うプログラムでは，使用する配列変数の個数について調べ，評価する必要があります。

tbasic では[*63]，100000000 個の配列変数を使用できますが，その 2 倍 200000000 は使えません。そこで，ここではそれぞれのプログラムで，どのくらいの素数表を作ることができるか比較してみましょう。

プログラム 2.1, 2.2, 4.1, 5.1 をもとにしたプログラムは簡単に評価できます。プログラム 2.1, 2.2 で n までの素数表を作るには，n 個の配列変数が必要です。ですから，篩える最大数は $10^8 = 100000000$ になります。プログラム 3.1($\mathtt{w_4}$) は n 個以上の配列変数が必要ですが，$\mathtt{w_4}$ であれば，追加は $\mathtt{mP_4} = 210$ 個程度なので，篩える最大数はやはり，10^8 です。またプログラム 4.1 では約 $2n$ 個の配列変数が必要ですから，篩える最大数は $10^8/2 = 50000000$ になります。更に，プログラム 5.1 では $n/2$ 個の配列変数が必要です。ですから，篩える最大数は $2 \cdot 10^8$ になります。また，プログラム 5.2, 5.3, 5.4 は，実際に，いくつか試してみることで，確かめることができます。これらを纏めてみると次のようになります。

プログラム	2.1′	2.2′	3.1($\mathtt{w_4}$)	4.1	5.1′	5.2($n=2$)	5.3($n=3$)	5.4($n=7$)
最大篩可能数	10^8	10^8	10^8	$5 \cdot 10^7$	$2 \cdot 10^8$	$3 \cdot 10^8$	$4 \cdot 10^8$	$7 \cdot 10^8$

[*60] プログラム 5.3 は表 Data を事前に計算してあるものですが，実行速度としては，プログラム 5.4 の $n = 3$ への書き直し版とほぼ同じです。

[*61] プログラム 4.1 ではメモリー不足のため 10^8 までの素数表は作成できません。ここに挙げた数値は 10^7 の 10 倍の数値です。メモリー制限がなければおよそこのくらいになると推定されます。

[*62] 車輪篩法と既約篩法は共に，$\mathtt{mP_n}$ と互いに素な対象を扱い，ある意味双対的（裏表的）な方法と言えます。

[*63] Ver.1.5 以上で Windows 10 での標準的環境の場合

プログラム 5.4 は，既約篩 $n=7$ のプログラムですが，$n=1,2,3,\ldots,8$ となるプログラムに簡単に書き直すことができます。しかし，表計算を行うため，$\mathtt{mP_n}$ の大きさにより，より多くの配列変数が必要になります。

既約篩法で篩に使用する配列変数は，既約篩を使わないときに比べて，$\phi(\mathtt{mP_n})/\mathtt{mP_n}$ となります。n が大きくなるにしたがって，より少ない配列変数で篩を実行することができます。他方，表計算に使用するものとして $3\mathtt{mP_n}+\phi(\mathtt{mP_n})$ 個の配列変数が篩表とは別に必要です。

この値をいくつか実際に計算してみると，

n	1	2	3	4	5	6	7	8	9
$\mathtt{mP_n}$	2	6	30	210	2310	30030	510510	9699690	223092870
$\phi(\mathtt{mP_n})$	1	2	8	48	480	5760	92160	1658880	36495360
$\phi(\mathtt{mP_n})/\mathtt{mP_n}$	0.5	0.333	0.267	0.229	0.208	0.192	0.181	0.171	0.164
$3\mathtt{mP_n}+\phi(\mathtt{mP_n})$	7	20	98	678	7410	95850	1623690	30757950	705773970

となります。

n が大きくなれば $\phi(\mathtt{mP_n})/\mathtt{mP_n}$ は小さくなります。$\mathtt{Rp_n}, \mathtt{Ir_n}, \mathtt{Floor_n}, \mathtt{Ceil_n}$ の計算速度はあまり変わりませんから，篩を実行する回数は少なくなり，計算は速くなります。しかし，その減少は大きくなるにつれて緩やかになり，n が大きくなることによる，効率化は余り大きくなりません。一方 $\mathtt{mP_n}$ は急激に大きくなり，その分だけの配列変数が必要になります。

実は，$n=9$ 以上の場合，表に必要な配列変数が膨大で，*64tbasic ではこの配列変数を確保できません。ですから，実際に実装可能なのは $n=8$ までです。しかし，$n=8$ の場合，表のために必要な配列変数は $3\cdot10^7$ を超えます。そのため，篩表に使う配列変数がその分少なくなります。

以下私の環境での既約篩法 $n=0,1,2,3,4,5,6,7,8$ での 10^8～$7\cdot10^8$ までの最大篩可能数の表を挙げます。

n	1	2	3	4	5	6	7	8
最大篩可能数	$2\cdot10^8$	$3\cdot10^8$	$4\cdot10^8$	$5\cdot10^8$	$6\cdot10^8$	$6\cdot10^8$	$7\cdot10^8$	$6\cdot10^8$

■まとめ

上での考察から，既約篩法プログラム 5.4($n=7$) が最も効率的で，高速という結果が得られます*65。プログラム 5.4 の弱点は多少プログラムが長いということですが，tbasic で本格的に大きな素数表を作る場合は，プログラム 5.4($n=7$) が適当でしょう。プログラム 5.4($n=8$) は，表計算に使用する配列変数が多く，既約篩に使う素数を増やすことで得られる篩の大きさの節約が及びません。ですから，今の場合，プログラム 5.4($n=8$) は，プログラム 5.4($n=7$) に及ばないと言えます。しかし，より大きな配列変数を使用できるプログラミング環境なら，プログラム 5.4($n=8$) または更に n が大きなプログラムを使用したほうが良いことも考えられます。n の選択は，使用できる配列変数の個数と，篩いたい大きさのバランスで考えるべきでしょう。

他方，プログラム 2.1′ は，プログラムが簡単というメリットがあります。また，余り大きくない素数表であれば十分高速に作成できます。例えば 10^5 程度までの素数表なら 1 秒も掛からず作成できます。ですから，10 万程度の素数表が必要な場合，プログラム 2.1′ で十分間に合います。特に，あるプログラムの中で，素数表が必要な場合，別に素数表を用意する代わりに小さな素数表作成プログラムとして内蔵させることもできます。このように，場合によっては，最も基本的なプログラム 2.1′ を使うのが良い場合もあります。

*64 少なくとも

*65 実際，プログラム 5.4($n=7$) で最大可能な $7\cdot10^8$ までの素数表を私の環境で，3分52秒 で作成できました。

6　区間を分割して篩う

前節までは 2 からある数 MaxN までの範囲を篩うというものでした。これらの方法では非常に大きな素数表を比較的短時間で作成することができました。しかし，これら方法では篩う表を格納するための MaxN 程度もしくはそれ以上の配列変数が必要でした。ですから，MaxN が大きな数の場合，表を格納する配列変数が確保できず篩が行えないという限界がありました。

ここでは，この限界を超えて，更に大きな素数表を作るための工夫を考えてみましょう。表を格納するための配列変数の確保の限界は一度に大きな素数表を作るということに起因していました。

ですから全体を作るのではなく，一部分を作るのなら，大きな配列変数を必要としない筈です。

例えば，10000001 から 10010000 までの素数というように，ある区間に限っての素数表を作ることを考えましょう。この場合，10000 個の配列変数があれば，篩ができると推測されます。ですから，この方法では，前節までの方法では作れなかった大きな数の素数表作ることができます。

この方法は篩う区間の制限だけですから，いくつかの注意をすれば，前節までの篩法の適用が可能です。

6.1　基本的な例

まず基本的なものとして，プログラム 2.1(基本篩)，プログラム 2.2(改良篩) での方法を考えましょう。

ここでは簡単のため，10001 から 20000 までの素数表を作るとします。10001 を篩う下限値，20000 を篩う上限値ということにします。配列変数 NT(1)〜NT(10000) を用意して，これらの数の素数性の情報を格納します。まず，10001 から 20000 までの数の表を作ります。

10001	10002	10003	10004	10005	10006	10007	10008	10009	10010
10011	10012	10013	10014	10015	10016	10017	10018	10019	10020
10021	10022	10023	10024	10025	10026	10027	10028	10029	10030
10031	10032	10033	10034	10035	10036	10037	10038	10039	10040
10041	10042	10043	10044	10045	10046	10047	10048	10049	10050
...									
19951	19952	19953	19954	19955	19956	19957	19958	19959	19960
19961	19962	19963	19964	19965	19966	19967	19968	19969	19970
19971	19972	19973	19974	19975	19976	19977	19978	19979	19980
19981	19982	19983	19984	19985	19986	19987	19988	19989	19990
19991	19992	19993	19994	19995	19996	19997	19998	19999	20000

(表 6.1)

ここで，ABase = 10000 と置くと，上の表は，

ABase + 1	ABase + 2	ABase + 3	ABase + 4	ABase + 5
ABase + 6	ABase + 7	ABase + 8	ABase + 9	ABase + 10
ABase + 11	ABase + 12	ABase + 13	ABase + 14	ABase + 15
ABase + 16	ABase + 17	ABase + 18	ABase + 19	ABase + 20
...				
ABase + 9981	ABase + 9982	ABase + 9983	ABase + 9984	ABase + 9985
ABase + 9986	ABase + 9987	ABase + 9988	ABase + 9989	ABase + 9990
ABase + 9991	ABase + 9992	ABase + 9993	ABase + 9994	ABase + 9995
ABase + 9996	ABase + 9997	ABase + 9998	ABase + 9999	ABase + 10000

(表 6.1)

と表されます。この表にある数 i に対して，NT(i − ABase) を対応させることで，それぞれ (表6.1) と NT(1)〜NT(10000) を対応させます。

次に，(表6.1) にある数をいくつかの素数で篩いますが，2.1 とは異なり，これらの篩う素数は (表6.1) にはありません。ですから，それらを予め別に用意する必要があります[*66]。今の場合，これらを篩う素数は，$\sqrt{20000} = 141....$ 以下の素数ですから，次のような表

2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139						

(素数表 1)

を予め用意します。実際のプログラムでは，もう一つ素数（今の場合は 149）が必要です[*67]。

これらの素数表は篩を使えばすぐに求めることができます。またプログラムでの利用は小さな素数表であれば，表を Data 文でプログラムに書き込み，それを Read 文で順次読み込むのが単純です。

この Data 文を作るには，篩のプログラムで必要なところまで素数表を作り，それを利用して作るのが良いでしょう。例えば，上の Data 文はプログラム 2.1′ に次のルーチンを加えることで，必要な行数の Data 文を作成することができます。

```
'-----------------------------------
'  Data 文作成ルーチン
'-----------------------------------
For i=2 To MaxN
   If NT(i)=0 then
      Counter = Counter + 1
      If Counter mod 10 = 0 then
         Print i
      Elseif Counter mod 10 = 1 then
         Print "Data "; i;",";
      Else
         Print i;",";
      End If
   End If
Next i
'-----------------------------------
```

ここでは，20000 より大きな数も篩えるように最大 100000 まで篩えるように $\sqrt{100000} = 316....$ の次の素数までの素数表の Data 文を作ります[*68]。

```
Data  2, 3, 5, 7, 11, 13, 17, 19, 23, 29
Data  31, 37, 41, 43, 47, 53, 59, 61, 67, 71
Data  73, 79, 83, 89, 97, 101, 103, 107, 109, 113
Data  127, 131, 137, 139, 149, 151, 157, 163, 167, 173
Data  179, 181, 191, 193, 197, 199, 211, 223, 227, 229
Data  233, 239, 241, 251, 257, 263, 269, 271, 277, 281
Data  283, 293, 307, 311, 313, 317
```

この Data 文をプログラムに張り付けて利用します。Data 文を Read 文を使って読み込む場合，素数データ格納のための配列変数を用意しなくても良いので，配列変数用メモリーの節約になります。ですからプログラムが大きくても良い場合は大量の Data 文を利用するという考えもあります。

[*66] この方法の限界はこの部分にあります。この用意する表は篩う区間に関わらず，篩われる数の大きさに依存しますから，この方法でも，また少ない範囲であっても，例えば，10^{100} から 10000 個の素数のような膨大な素数の表は作れません。

[*67] 次の素数までの表を作る理由は，p * p <= UBound の脱出判定のために，実際には篩わない次の素数が必要だからです。

[*68] 具体的には，プログラム 2.1' に上記の Data 文作成ルーチンを追加して，例えば MaxN = 320 としてプログラムを動作させます。

例えば，最大 10^{10} までの数の素数を求めるために区分篩法を使うとすると，10^5 までの素数表を用意する必要があります。10^5 までの素数は全部で 9592 個ありますが，これらをこの方法で上のように 1 行に 10 個の素数を書くとすると，960 行のデータ文で可能になります。

次に，篩い始める下限値 LBound と篩終わりの上限値 UBound を決めます。今の場合，10001 から 20000 まで篩うので，LBound = 10001，UBound = 20000 と置きます。

(素数表 1) にある素数 p に対して，LBound と UBound の間で篩を実行します。その場合の乗数 m は

$$\frac{\mathrm{LBound}}{p} \leqq m \leqq \frac{\mathrm{UBound}}{p}$$

の範囲にありますが，m は整数なので，

$$\mathtt{Ceil}\left(\frac{\mathrm{LBound}}{p}\right) \leqq m \leqq \mathtt{Floor}\left(\frac{\mathrm{UBound}}{p}\right)$$

の範囲にありますが，p 以上の数ですから，即ち，

$$m = \mathtt{Max}\left(p, \mathtt{Ceil}\left(\frac{\mathrm{LBound}}{p}\right)\right), \ \ldots, \ \mathtt{Floor}\left(\frac{\mathrm{UBound}}{p}\right)$$

となります。ここで，Ceil は小数切り上げ関数，即ち，Ceil(x) は，x 以上の最小整数を与える関数です。また Floor は小数切り捨て関数，即ち，Floor(x) は，x 以下の最大整数を与えます[*69]。即ち，StartN = $\mathtt{Max}\left(p, \mathtt{Ceil}\left(\frac{\mathrm{LBound}}{p}\right)\right)$，StopN = $\mathtt{Floor}\left(\frac{\mathrm{UBound}}{p}\right)$ とし，

$$m = \mathtt{StartN},\ \mathtt{StartN}+1,\ \ldots,\ \mathtt{StopN}$$

の範囲を篩います。（改良篩の場合も，$p = 2$ のときは，これらの m を乗数として篩います。）

プログラム 2.1(基本篩) の方法でこの部分の処理を表すと，

```
'----------------------
StartN = Max(p,Ceil(LBound/p))
StopN = Int(UBound/p)
For i=StartN To StopN
   NT(i*p - ABase)=1
Next i
'----------------------
```

となります。

また，プログラム 2.2(改良篩) では，p が 3 以上の場合，乗数 m は奇数のみで良かったので，Ceil $\left(\frac{\mathrm{LBound}}{p}\right)$ が奇数ならば，StartN = Max(Ceil $\left(\frac{\mathrm{LBound}}{p}\right)$)，偶数ならば，StartN = Max(Ceil $\left(\frac{\mathrm{LBound}}{p}\right)$) + 1 として，

$$m = \mathtt{StartN},\ \mathtt{StartN}+2,\ \mathtt{StartN}+4,\ \ldots\ ,\ \mathtt{StopN}\ \ \mathtt{Step2}$$

の範囲を一つおき step 2 で篩います。

プログラム 2.2(改良篩) の方法でこの部分の処理を表すと，

```
'----------------------
'  p は 3 以上
   StartN = Max(p,Ceil(LBound/p))
   If StartN mod 2 = 0 then StartN = StartN + 1
```

[*69] Floor 関数は，tbasic では Int 関数と同じです。また，ガウスの記号 $[x]$ とも同じです。

```
   StopN = Int(UBound/p)
   For i=StartN To StopN step 2
      NT(i*p - ABase)=1
   Next i
'----------------------
```

となります。

以上のことを注意して，プログラム 2.2 を書き換えると次が得られます。

プログラム **6.1**

```
'----------------------
' エラトステネスの篩：区分改良篩
'    10001 から 20000 までの素数を見つける
'    10000 個の配列変数 NT を使う
'    sqr(100000)= 316... より 316 より大きな次の素数 317 までの素数表を用意する。
'----------------------
Dim NT(10000)
Const LBound = 10001
UBound = LBound+10000-1
ABase= LBound-1
' 素数表 以下の奇素数で篩う
Data  2, 3, 5, 7, 11, 13, 17, 19, 23, 29
Data  31, 37, 41, 43, 47, 53, 59, 61, 67, 71
Data  73, 79, 83, 89, 97, 101, 103, 107, 109, 113
Data  127, 131, 137, 139, 149, 151, 157, 163, 167, 173
Data  179, 181, 191, 193, 197, 199, 211, 223, 227, 229
Data  233, 239, 241, 251, 257, 263, 269, 271, 277, 281
Data  283, 293, 307, 311, 313, 317
'----------------------
Read p
StartN = Max(p,Ceil(LBound/p))
StopN = Int(UBound/p)
For i=StartN To StopN
   NT(p*i - ABase)=1
Next i
Read p
While p*p<=UBound
   StartN = Max(p,Ceil(LBound/p))
   If StartN mod 2 = 0 then StartN = StartN + 1
   StopN = Int(UBound/p)
   For i=StartN To StopN step 2
      NT(p*i - ABase)=1
   Next i
   Read p
End While
For i=LBound To UBound
   If NT(i-ABase)=0 then Print i
Next i
End

'_____________________
```

このプログラムは $\sqrt{100000} = 316....$ を超える素数の表を持っているので，10^5 以下の数を篩うことができます。ですから，Const LBoundN = 10001 の 10001 の部分を，20001，30001，...，90001 等に書き換えることで，LBoundN から LBoundN + 10000 の間にある素数の表を作ることができます。[*70]

[*70] Const LBoundN = 1 とすると，1 は篩われず，篩の後，NT(1) = 0 となりますので，少し注意が必要です。

6.2 既約篩法 $\mathtt{mP}_n$ を行った場合

区分篩は既約篩法にも適用可能です。

例えば，$\mathtt{mP}_2 = 6$ として，1001 から $\mathtt{mP}_2$ と互いに素な整数を 100 個並べてみると，

1001	1003	1007	1009	1013	1015	1019	1021	1025	1027
1031	1033	1037	1039	1043	1045	1049	1051	1055	1057
1061	1063	1067	1069	1073	1075	1079	1081	1085	1087
1091	1093	1097	1099	1103	1105	1109	1111	1115	1117
1121	1123	1127	1129	1133	1135	1139	1141	1145	1147
1151	1153	1157	1159	1163	1165	1169	1171	1175	1177
1181	1183	1187	1189	1193	1195	1199	1201	1205	1207
1211	1213	1217	1219	1223	1225	1229	1231	1235	1237
1241	1243	1247	1249	1253	1255	1259	1261	1265	1267
1271	1273	1277	1279	1283	1285	1289	1291	1295	1297

(6.2 篩表 2)

となります。これらを 100 個の配列変数 $\mathtt{NT}(1)$〜$\mathtt{NT}(100)$ に対応させます。上の (6.2篩表2) に $\mathtt{Ir}_2$ を施すと[*71]，

334	335	336	337	338	339	340	341	342	343
344	345	346	347	348	349	350	351	352	353
354	355	356	357	358	359	360	361	362	363
364	365	366	367	368	369	370	371	372	373
374	375	376	377	378	379	380	381	382	383
384	385	386	387	388	389	390	391	392	393
394	395	396	397	398	399	400	401	402	403
404	405	406	407	408	409	410	411	412	413
414	415	416	417	418	419	420	421	422	423
424	425	426	427	428	429	430	431	432	433

ですから，$\mathtt{ABase} = 333$ として，(6.2篩表2) にある j に対して，$\mathtt{NT}(\mathtt{Ir}_2(j) - \mathtt{ABase})$ を対応することで，(6.2篩表2) を $\mathtt{NT}(1)$〜$\mathtt{NT}(100)$ 対応させることができます。

一般に，

$\mathtt{MaxN}$ 個の配列変数 $\mathtt{NT}(1)$〜$\mathtt{NT}(\mathtt{MaxN})$ を使うとします。このとき，篩表

$r_{\mathtt{ABase}+1}$	$r_{\mathtt{ABase}+2}$	$r_{\mathtt{ABase}+3}$	$r_{\mathtt{ABase}+4}$	$r_{\mathtt{ABase}+5}$
$r_{\mathtt{ABase}+6}$	$r_{\mathtt{ABase}+7}$	$r_{\mathtt{ABase}+8}$	$r_{\mathtt{ABase}+9}$	$r_{\mathtt{ABase}+10}$
$r_{\mathtt{ABase}+11}$	$r_{\mathtt{ABase}+12}$	$r_{\mathtt{ABase}+13}$	$r_{\mathtt{ABase}+14}$	$r_{\mathtt{ABase}+15}$
$r_{\mathtt{ABase}+16}$	$r_{\mathtt{ABase}+17}$	$r_{\mathtt{ABase}+18}$	$r_{\mathtt{ABase}+19}$	$r_{\mathtt{ABase}+20}$
$r_{\mathtt{ABase}+21}$	$r_{\mathtt{ABase}+22}$	$r_{\mathtt{ABase}+23}$	$r_{\mathtt{ABase}+24}$	$r_{\mathtt{ABase}+25}$
$r_{\mathtt{ABase}+26}$	$r_{\mathtt{ABase}+27}$	$r_{\mathtt{ABase}+28}$	$r_{\mathtt{ABase}+29}$	$r_{\mathtt{ABase}+30}$
$r_{\mathtt{ABase}+31}$	$r_{\mathtt{ABase}+32}$	$r_{\mathtt{ABase}+33}$	$r_{\mathtt{ABase}+34}$	$r_{\mathtt{ABase}+35}$
$r_{\mathtt{ABase}+36}$	$r_{\mathtt{ABase}+37}$	$r_{\mathtt{ABase}+38}$	$r_{\mathtt{ABase}+39}$	$r_{\mathtt{ABase}+40}$
$r_{\mathtt{ABase}+41}$	$r_{\mathtt{ABase}+42}$	$r_{\mathtt{ABase}+43}$	$r_{\mathtt{ABase}+44}$	$r_{\mathtt{ABase}+45}$
$r_{\mathtt{ABase}+46}$	$r_{\mathtt{ABase}+47}$	$r_{\mathtt{ABase}+48}$	$r_{\mathtt{ABase}+49}$	$r_{\mathtt{ABase}+50}$
...				
.	.	.	.	.
.	.	.	.	$r_{\mathtt{ABase}+\mathtt{MaxN}}$

(6.2 篩表 n)

[*71] 関数 $\mathtt{Rp}_2(i) = r_i$，$\mathtt{Ir}_2(s)$ は定義 5.1 の通りとします。

に含まれる $\mathrm{Rp_n}(\mathrm{ABase}+i) = r_{\mathrm{ABase}+i}$ に対して，$\mathrm{NT}(i)$ を対応させます。即ち，(6.2篩表n) に含まれる r に対して，$\mathrm{NT}(\mathrm{Ir_n}(r) - \mathrm{ABase})$ が対応します。

$r_{\mathrm{ABase}+1}$ から $r_{\mathrm{ABase}+\mathrm{MaxN}}$ までが篩う対象数です。そこで，篩う下限値を $\mathrm{LBound} = r_{\mathrm{ABase}+1}$，篩う上限値を $\mathrm{UBound} = r_{\mathrm{ABase}+\mathrm{MaxN}}$ と表します。また，篩う区間幅は $\mathrm{LInterval} = \mathrm{UBound} - \mathrm{LBound} + 1$ となります。即ち，

$$\begin{array}{lcl} \mathrm{LBound} & = & \mathrm{Rp_n}(\mathrm{ABase}+1) \\ \mathrm{UBound} & = & \mathrm{Rp_n}(\mathrm{ABase}+\mathrm{MaxN}) \\ \mathrm{ABase} & = & \mathrm{Ir_n}(\mathrm{LBound}) - 1 \\ \mathrm{MaxN} & = & \mathrm{Ir_n}(\mathrm{UBound}) - \mathrm{ABase} \\ \mathrm{LInterval} & = & \mathrm{Rp_n}(\mathrm{ABase}+\mathrm{MaxN}) - \mathrm{Rp_n}(\mathrm{ABase}+1) + 1 \end{array} \qquad \text{(6.2 関係式)}$$

となります。

上の例 (6.2篩表2)（$\mathrm{mP_2} = 6$）の場合は，$\mathrm{LBoundN} = 1001$，$\mathrm{UBoundN} = 1297$，$\mathrm{ABase} = 333$，$\mathrm{MaxN} = 100$，$\mathrm{LInterval} = 297$ です。

篩は，LBoundN～UBoundN の間を素数 p で篩います。篩う素数は，(6.2篩表n) の中にありませんから，6.1 と同様に，別に用意する必要があります。素数の用意の仕方はいくつか考えられますが，ここでは，擬似コードを書くために次の定義をしましょう。

定義 6.1. 篩う最初の素数を与える関数を firstPrime とし，次の素数を与える関数を nextPrime と表す。

より正確には，firstPrime は $\mathrm{mP_n}$ に対して，$n+1$ 番目の素数 p_{n+1} を与えます。また，nextPrime は firstPrime または nextPrime によって与えられた直前の素数を p_m とするとき，p_{m+1} を与えます。

firstPrime または nextPrime によって与えられた素数 p に対して，LBound と UBound の間で篩を実行します。その場合の乗数 m は

$$\frac{\mathrm{LBound}}{p} \leqq m \leqq \frac{\mathrm{UBound}}{p}$$

の範囲にあります。ここで m は整数なので，

$$\mathrm{Ceil}\left(\frac{\mathrm{LBound}}{p}\right) \leqq m \leqq \mathrm{Floor}\left(\frac{\mathrm{UBound}}{p}\right)$$

となり，更に m は (4.2篩表n) の中にある数[*72]ですから，

$$\mathrm{Ceil_n}\left(\mathrm{Ceil}\left(\frac{\mathrm{LBound}}{p}\right)\right) \leqq m \leqq \mathrm{Floor_n}\left(\mathrm{Floor}\left(\frac{\mathrm{UBound}}{p}\right)\right)$$

の範囲にあります。一方 m は p 以上ですから，

$$\mathrm{Max}\left(p, \mathrm{Ceil_n}\left(\mathrm{Ceil}\left(\frac{\mathrm{LBound}}{p}\right)\right)\right) \leqq m \leqq \mathrm{Floor_n}\left(\mathrm{Floor}\left(\frac{\mathrm{UBound}}{p}\right)\right)$$

の範囲にあります。

即ち，

$$\mathrm{Ir_n}\left(\mathrm{Max}\left(p, \mathrm{Ceil_n}\left(\mathrm{Ceil}\left(\frac{\mathrm{LBound}}{p}\right)\right)\right)\right) \leqq \mathrm{Ir_n}(m) \leqq \mathrm{Ir_n}\left(\mathrm{Floor_n}\left(\mathrm{Floor}\left(\left(\frac{\mathrm{UBound}}{p}\right)\right)\right)\right)$$

[*72] 即ち，$\mathrm{mP_n}$ と互いに素な数

の範囲にあります。従って

$$\mathrm{iLBound} = \mathrm{Ir_n}\left(\mathrm{Max}\left(p, \mathrm{Ceil_n}\left(\mathrm{Ceil}\left(\frac{\mathrm{LBound}}{p}\right)\right)\right)\right),\ \mathrm{iUBound} = \mathrm{Ir_n}\left(\mathrm{Floor_n}\left(\left[\frac{\mathrm{UBound}}{p}\right]\right)\right)$$

とするとき，

$$i = \mathrm{iLBound},\ \ldots\ , \mathrm{iUBound}$$

に対して，

$$m = \mathrm{Rp_n(i)}$$

が乗数になります。即ち，

$$\mathrm{Rp_n(i)} \cdot p$$

が篩われる数になります。またこれに対応する NT は，

$$\mathrm{NT}(\mathrm{Ir_n}(\mathrm{Rp_n(i)} \cdot p) - \mathrm{ABase})$$

になります。

以上のことに注意して，5.3 擬似コード 2 を区分既約篩版に書き直すと次が得られます。

```
'-----------------------------------
' 擬似コード 2'(区分既約篩版)
'-----------------------------------

p = firstPrime
While p * p <= UBound
      iLBound = Ir_n(Max(p, Ceil_n(Ceil(LBound/p))))
      iUBound = Ir_n(Floor_n(Int(UBound/p)))
      For i = iLBound to iUBound
            NT(Ir_n(Rp_n(i) * p) - ABase) = 1
      Next i
      p = nextPrime
End While

'-----------------------------------
```

6.3 プログラム例

ここでは，前項の擬似コード 2' を使って具体的なプログラムを作りましょう。

課題

$1 \leqq i \leqq 100$ となる整数 i に対して $(i-1)\cdot 10^8+1$ から $i\cdot 10^8$ までの素数表を作成する。

このプログラムを $i = 1, \ldots, 100$ と実行してその結果を纏めれば，10^{10} 迄の素数表が作成できます。

[ステップ 1] $\mathtt{mP_n}$ の決定と設定

tbasic で現実的に利用可能で，最も効率的なのは $n = 7$ での既約篩法でした。ここでは比較的大きな数で，大きな素数表を作るということで，プログラム 5.4 をもとにして，$n = 7$ での既約篩法を使うことにします。

主な定数，宣言はプログラム 5.4 のものをそのまま使います。このとき，$\mathtt{Ceil_7}$ だけ，追加する必要がありますが，これは $\mathtt{Floor_7}$ と同様に，cE 作成擬似プログラムを利用して作ります。

また，i の部分を For 文で囲むことで，いくつかの区間を連続的に実行できるようにするものとします。

[ステップ 2] 篩用配列変数の準備

いくつかの変数値を決める必要があります。

$i = 1, 2, \ldots 100$ に対し[*73]，$\mathtt{tLBound} = (\mathtt{i}-1)\cdot \mathtt{10}^8+1$，$\mathtt{tUBound} = \mathtt{i}\cdot \mathtt{10}^8$ と置きます。篩う区間の幅は 10^8 で i に関係しません。

まず，問題の条件から篩う下限は $\mathtt{LBound} = \mathtt{Ceil_7}(\mathtt{tLBound})$，篩う上限は $\mathtt{UBound} = \mathtt{Floor_7}(\mathtt{tUBound})$ です。さらに，(6.2関係式) を使うと，$\mathtt{ABase} = \mathtt{ir_7}(\mathtt{LBound}) - 1$，$\mathtt{MaxN} = \mathtt{ir_7}(\mathtt{UBound}) - \mathtt{ABase}$ が得られます。元々の篩う区間幅は 10^8 ですが，i の値によって，これらの値，特に MaxN の値は少し変化します。For 文を使って計算してみると，$i = 1, 2, \ldots, 100$ ときの最大値は 18052540 であることが分かります。そこで，NT の宣言は最大値 18052540 とし，Dim NT(18052540) と宣言します。実際の篩は $\mathtt{MaxN} \leqq \mathtt{18052540}$ となる MaxN まで使用します。

課題で例えば $i = 50$ の場合は，4900000000 から 5000000000 までの素数表を作ることになりますが，実際の篩う値は，

LBound = 4900000003,
UBound = 4999999997,
LInterval = 99999995,
ABase = 884574250,
MaxN = 18052534

となります。

[ステップ 3] 素数表の準備

区分法の場合，篩う最大数の平方根までの素数表を用意する必要があります。今の場合，10^{10} まで篩うことになりますから，$\sqrt{10^{10}} = 10^5$ の以下の素数表が必要です。10^5 以下の素数は全部で 9592 個ありますが，ここでは脱出条件のために素数を一つ増やして，9593 個の素数表を用意します。

[*73] 実際のプログラムでは i の代わりに，ii を使うことにします。

素数表を用意してそれを利用する方法は，いくつか考えられますが，ここでは Data 文としてプログラムに書き込みむのではなく，配列変数 Primes(1)～Primes(9593) を用意しそれに読み込むことにしましょう。素数表はプログラムの中で作成し，その結果を配列変数 Primes に読み込みます。

素数表の作成は今の場合，$n = 7$ での既約篩法の設定をしますから，それを利用して作ることにします。10^5 を超える最初の素数 100003 までの素数表の作成をプログラム 5.4 の方法で行います。篩う最大数を $10^5 + 3$ とするだけです。その後，篩い終わった配列変数 NT から，素数データを Primes に読み込むのは，$2, 3, 5, 7, 11, 13, 17$ を設定した後，NT(i) = 0 となる i を設定します。例えば，

```
'-----------------------------------
Primes(1)=2: Primes(2)=3:Primes(3)=5: Primes(4)=7
Primes(5)=11: Primes(6)=13:Primes(7)=17
Counter = 7
For i=2 To Irn(FloorN(10^5))
   If NT(i)=0 then
      Counter= Counter +1
      Primes(Counter)=Rpn(i)
   End If
Next i
'-----------------------------------
```

で実現されます。ここで，Primes(1)～Primes(7) は実際の篩では使用されませんから，この設定はしなくても構いません。

[ステップ 4]firstPrime，nextPrime 関数の準備

素数表用配列変数 Primes が上で与えられていますから，これを使って firstPrime，nextPrime 関数を定めます。

今の場合 mP_7 ですから，firstPrime は Primes(8) になります。また，nextPrime は変数 primeCounter を用意し，firstPrime を呼んだとき，primeCounter = 8 と初期化し，nextPrime を呼んだとき，primeCounter を 1 だけ増やし，Primes(primeCounter) を返します。これらの関数の設定は具体的には次で実現されます。

```
'-----------------------------------
Public primeCounter

Function firstPrime
   primeCounter = 8
   firstPrime = Primes(primeCounter)
End Function

Function nextPrime
   primeCounter = primeCounter + 1
   nextPrime = Primes(primeCounter)
End Function
'-----------------------------------
```

[ステップ 5] 篩

篩は，擬似コード 2'(区分既約篩版) をほぼそのままで使用します。

tbasic では数値配列変数の初期値は 0 でした。ですから，NT(i) の初期値は 0 です。一度だけ実行する場合は，これを仮定できますが，篩う区間を変えて繰り返し実行する場合は，明示的に初期設定する必要があります。

ここでは，今回篩いに使う NT(i) の範囲 $i = 1, \ldots$ MaxN の初期化を行います。これは，以下の Sub を Call することで実現します。

```
'-----------------------------------
Sub InitNT(MaxN)
   For i=1 To MaxN
      NT(i)=0
   Next i
End Sub
'-----------------------------------
```

[ステップ 6] 結果の処理等

篩を行った結果を利用する目的は色々考えられます。素数表をファイルに保存することも考えられますが，ここでは取り合えず，素数の個数を求めることにします。以下のプログラムで実現されますが，

```
'-----------------------------------
Counter = 0
For i=1 To MaxN
   If NT(i)=0  Then
      Counter= Counter +1
Next i
Print Counter
'-----------------------------------
```

但し課題で，$i = 1$（即ち，篩う範囲が 1〜10000000）の場合は注意が必要です。それは，まず，この場合，NT(1) = 0 となり，この場合が個数に含まれます[*74]。また，$n = 7$ での既約篩法では，$p = 2, 3, 5, 7, 11, 13, 17$ は素数表をにはありませんので，このことを考慮しなくてはなりません。$i > 1$ の場合は，このようなことはありません。

[*74] 実際，区分篩でない場合，NT(1) は使用しませんでした。

以上を纏めるとプログラムが出来上がります。少し大きいプログラムですが，プログラム全体を挙げます。

プログラム **6.2**

```
'--------------------------------
' 区分既約篩法 n=7：
' (ii-1)*10^8+1 から ii*10^8
' ii=1,...,100
' までの素数表の作成
'--------------------------------
' 2,3,5,7,11,13,17(510510) と互いに素な表を使って篩う
Public Const mPn=510510
Public Const phiPn=92160:' 1*2*4*6*10*12*16 = phi(510510)
Public Const MaxPD=9593
Public MaxSPN: MaxSPN= 10^5+3
Public rP(phiPn), iR(mPn),  fL(mPn), cE(mPn)
Public Primes(MaxPD)
Public NT(18052540)
Public primeCounter

Call MakeArrayData
Call MakeSievePrimeTable

' For ii=1 To 100

ii =2

DLBound = (ii-1)*10^8+1
DUBound = ii*10^8
LBound = CeilN(DLBound)
UBound = FloorN(DUBound)
LInterval = UBound - LBound + 1
ABase = Irn(LBound)-1
MaxN = Irn(UBound)-ABase

Call InitNT(MaxN)

Print Time$;" Sieve Start"
p=firstPrime
While p*p <= UBound
'   Print p;"-Sieving"
   iLBound = Irn(Max(p,CeilN(Ceil(LBound/p))))
   iUBound = Irn(FloorN(Int(UBound/p)))
   For i=iLBound To iUBound
      NT(Irn(Rpn(i)*p)-ABase)=1
   Next i
   p=nextPrime
End While
Print Time$;" Sieve End"

Counter = 0
For i=1 To MaxN
   If NT(i)=0  Then
      Counter= Counter +1
   End If
Next i
Print ii;" : ";Counter

' Next ii
End
```

```
Function Rpn(i)
   Rpn = mPn* Int(i/phiPn) + rP(i mod phiPn)
End Function

Function Irn(j)
   Irn = phiPn*Int(j/mPn) + iR(j mod mPn)
End Function

Function FloorN(i)
   FloorN= i +fL(i mod mPn)
End function

Function CeilN(i)
   CeilN= i +cE(i mod mPn)
End function

Sub MakeArrayData
Print Time$;
Print "making rP Data"
   Counter = 0
   rP(0)=-1
   For i=1 To mPn
      If GCD(mPn,i)=1 then
         Counter = Counter + 1
          rP(Counter)=i
      End If
   Next i
Print Time$;
Print "making iR Data"
   For i=1 To phiPn
      iR(rP(i))=i
   Next i
Print Time$;
Print "making fL Data"
   fL(0)=-1
   For i=1 To mPn-1
      If Gcd(i,mPn)=1 then
         fD=0
      Else
         fD=fD -1
      End If
      fL(i)=fD
   Next i
Print Time$;
Print "making cE Data"
   For i = 1 To mPn - 1
   If Gcd(mPn - i, mPn) = 1 then
      cD = 0
   Else
      cD = cD + 1
   End If
   cE(mPn - i) = cD
   cE(0)=1
Next i
End Sub

Sub MakeSievePrimeTable
Print "making prime data for sieving"
 i=2
```

```
 p=Rpn(2)
 sMaxN=MaxSPN
 While p*p<= sMaxN
    iMax = Irn(FloorN(Int(sMaxN/p)))
    For i0=i To iMax
       NT(Irn(Rpn(i0)*p))=1
    Next i0
    i=i+1
    While NT(i)=1:i=i+1:End While
    p=rpn(i)
 End While
 Primes(1)=2: Primes(2)=3:Primes(3)=5: Primes(4)=7
 Primes(5)=11: Primes(6)=13:Primes(7)=17
 Counter = 7
 For i=2 To Irn(FloorN(MaxSPN))
    If NT(i)=0 then
       Counter= Counter +1
       Primes(Counter)=Rpn(i)
    End If
 Next i
End Sub

Function firstPrime
   primeCounter = 8
   firstPrime = Primes(primeCounter)
End Function

Function nextPrime
   primeCounter = primeCounter + 1
   nextPrime = Primes(primeCounter)
End Function

Sub InitNT(MaxN)
   For i=1 to MaxN
      NT(i)=0
   Next i
End Sub
'-----------------------------------
```

*[75]

*[75] このプログラムは約 40 秒で 1 回の実行が可能でした。

7　付録：大きな素数表を作る

エラトステネスの篩は素数表を効率的に作成する優れた方法です。現在では，大きな素数表を必要とする理由は余り多くありませんが，それでも大きな素数表を作ってみたいと思うことがあるかもしれません。

勿論，時間制限なしで，計算機の資源・速度が十分にある環境なら，基本篩法で十分大きな素数表ができるでしょう。ここでは，そのような特別な環境ではなく，身近な環境の下で実行するという条件で，この問題を少し考えてみましょう[*76][*77]。

7.1　tbasic からの移植

前節で説明したプログラム 6.2 は 40 秒程度で一回が実行されます[*78]。これで幅が 10^8 の素数表が作成できます。またこのプログラムを少し変更することで 10^{10} 以下の任意の 10^8 幅の素数表を作成できます。ですから，これを 100 回実行すれば 2 時間以内で 10^{10} までの素数表は作れるでしょう。また，64 ビット windows 上では tbasic は並行的に実行できます[*79]。そうすれば 1 時間以内に 10^{10} までの素数表は作成できるでしょう。更にこれ以上の時間を掛ければ tbasic での計算精度の範囲でもっと大きな素数表が作成できる筈です。

しかし，tbasic はインタプリタですから，元々このような計算には適していません。

そこで身近にある環境で，もう少し計算に適したものを利用して調べてみましょう。ここでは，Visual Studio 2019 での Visual Basic と C# を使うことにします。[*80]ここでの目標は，10^{13} 程度の素数表を作成するとして，時間・資源等の必要性を評価します[*81]。

7.1.1　Visual Studio への移植

Visual Studio では色々な言語を扱うことができますが，tbasic からの移植の容易さで，まず，Visual Basic への移植を考えます。更に，広く使われ，高速に動作すると言われている C# に Visual Basic のプログラムを移植することにします。

Visual Studio は高度な機能を持ったオブジェクト指向言語システムで，多くの専門的なプログラムも書かれています。これに対して tbasic は個人用に使いやすいツールとしての位置づけで，tbasic のプログラムは小規模で機能の限定されたものです。ですから，tbasic のプログラムを Visual Studio のようなシステム上への移植の目的は，プログラム実行の高速化にあります。

ですから，Visual Studio の高度な機能を使うのではなく，最も簡単に移植し，高速化を図るものとします。オブジェクト指向言語の特徴である class のようなものは使用しません。

そのため，最低限の Windows フォームアプリケーションとして作成することにします[*82]。Windows フォームアプリケーションといっても tbasic でのプログラムを実行するためのものですから，フォームについては，

[*76] この節での説明は追加的なメモとしての位置づけですので，付録とします。

[*77] 身近な環境という意味は私の環境，エントリーと思えるデスクトップマシンで，OS は 64 ビット Windows 10 です。ハイエンドマシンで実行すれば数倍高速でしょう。また以下の記述は 2020 年 6 月時点でのものですから，計算機の機能向上で実行時間等の短縮はあるでしょう。

[*78] これまで通り私の環境での結果です。

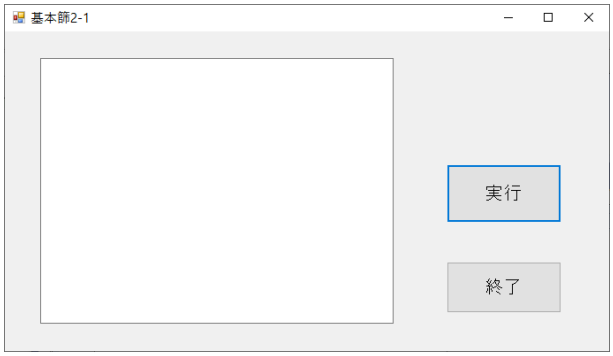
[*79] 私の環境では 3 つの tbasic を同時に立ち上げて実行でき，それぞれ速度も余り変わりません。

[*80] Visual Studio Community は個人は，無料で使用できます。別なコンパイラを使っても似たようなことができると思います。

[*81] 10^{13} としたのは，この大きさの素数表を，PC で作成したとの報告が，インターネット上でいくつかあるからです。スーパーコンピューター等の特殊な計算機を用いれば勿論もっと大きな素数表が短時間で作成できるでしょう。

[*82] コンソールアプリケーションとしての移植も考えられますが，特に高速にはなりません。

一つのフォームを用意し，その上に実行ボタン，終了ボタン，結果表示用の TextBox を配置するだけです。これは Visual Stadio の IDE 上で簡単に作成できます。つまり，以下のような画面を作成します。



ここで，実行ボタンを押すと，計算を実行し，その結果を左の TextBox に表示します。終了ボタンを押すとプログラムが終了します。これが今回の tbasic プログラムの移植の舞台です。

また，Visual Studio では，プロジェクト，コンパイルの項でターゲット CPU を x64 と指定することで，64 ビットアプリケーションをを作成できます。64 ビットアプリケーションは 32 ビットアプリケーションよりも高速になりますので，ここではこのオプションを使ってコンパイルすることにします。

実行時間の計測は「デバッグなしで開始」で行います。

7.1.2 Visual Basic への移植

tbasic と Visual Basic は基本的な部分の文法は似ていますから，Visual Basic を少し学んだことのある人にとって，tbasic のプログラムの移植は簡単です。

Visual Basic と tbasic の大きな違いは，変数宣言に対する考え方です。tbasic では，基本的に変数は暗黙宣言（特に変数を宣言しなくて使う。）ですが，Visual Basic では，変数は宣言して使うのが推奨です[*83]。

変数を宣言して使うことの利点は，変数の型による計算の特徴を生かすことができるからです。

例えば，tbasic では数と言えば一種類 10 バイト（80 ビット）浮動小数点数[*84]ですが，Visual Basic で数は，何種類もあります。浮動小数点数でも 32 ビット Single，64 ビット Double の 2 種類があります。また整数型には，32 ビット Integer，64 ビット Long があります。それぞれ計算の精度，速度が異なりますから，計算の目的によって選ぶ必要があります[*85]。今回はエラトステネスの篩という比較的単純な計算ですから，計算全体について，使う型の検討が可能です。

また詳細コンパイルオプションでは，「整数オーバーフローのチェックを削除」，「最適化を有効化する」にチェックを入れます。これによってさらに高速化されます。

7.1.3 C# への移植

C# は，Basic とは演算記号や，文の指定の仕方は異なりますが，同じ手続き系の言語ですので，構文等は対応するものがあります。ですから，今回移植する程度のプログラムでの計算式等の書き換えは比較的容易です。

[*83] 暗黙宣言も使えますが，すべて宣言するモードで使うのが良いでしょう。全て宣言して使うモードに設定するには，プロジェクトのプロパティ，コンパイルの項で Option Strict On にします。

[*84] Extended と呼ばれるものです。

[*85] tbasic では，数は一種類ですが，かなり精度が高いので計算の種類に関わらず，使用できますが，その分速度が幾分遅くなります。

7.2 篩法

上に挙げた篩法のプログラムをそれぞれ，Visual Basic と C# に移植して速度を比較してみましょう。

Visual Studio 2019 で扱える配列数は最大で 0X7FFFFFC7 = 2147483591 とされています[*86]。特に，配列のインデックスとして使う数の型は 32 ビット整数 Integer です。一方，10^{10} は 32 ビット整数ではありませんから，これ以上の数を扱う場合は 64 ビット整数 Long を使う必要があります。

基本篩，改良篩，車輪篩では，篩う最大数までの配列が必要ですから，上の配列数の制限からこれらの篩法では一度に最大 $2 \cdot 10^9$ 程度迄しか篩うことはできません。これらの数は 32 ビット整数ですから，この範囲では，32 ビット整数で計算可能です。

しかし，既約篩法を使うとそれより多くの数を篩うことができます。例えば，$n = 6$ の場合は 5 倍程度大きな数を篩えますから，10^{10} 迄程度まで一度に篩うことができるでしょう。一方 10^{11} 迄の素数の個数は 4118054813 で，0X7FFFFFC7 を超えますから，10^{11} 迄一度に篩うことはできません。ですから，10 の冪で考えれば，一度に篩える数は最大 10^{10} までです。そしてこれ以上の素数表を作る場合は区分法を利用しなければなりません。

10^{11} 以上の素数表の作成に区分法を用いる場合，どのような篩法を使っても，篩う数の計算は 10^{11} 以上の計算を行う必要があります。この計算は 64 ビット整数 Long を使う必要があります。ここでの目標は 10^{13} までの素数表の作成ですから，

- 整数計算は 64 ビット整数で行う。
- 配列のインデックス 32 ビット整数を使う。

が基本となります。

7.2.1 Visual Basic での比較

上で挙げた tbasic のプログラムを Visual Basic に移植してみます。この文書は Visual Basic の説明のための文書ではありませんので，説明は概略です。[*87]

(1) 基本篩法

基本篩法はほぼプログラム 2.1'（基本篩）をそのまま使えますが，変数宣言と Print 等は調整する必要があります。

Visual Studio では NT は特別な意味があるので，ここでは，篩用の配列変数 NT の名前を変更して，arrayNT とします。tbasic では，配列は数としての宣言でしたが，Visual Studio ではバイト配列が使えるので，配列はバイトとして宣言します[*88]。宣言は，Form の冒頭に置くことにして，アクセス修飾子を付けて

```
Private Const MaxN As Integer = 1000000000 ' 10^9
Private arrayNT(MaxN) As Byte
```

とします。それ以外の変数は，Sub の中で宣言します。更に，64 ビット整数 Long から 32 ビット整数 integer への変換関数として，CInt を使います。

Print 文は Visual Basic には無いので，出力用 TextBox への出力 Sub を作成します。文字列だけを出力す

[*86] これはバイト配列の場合で，一般の場合はこれよりも，少なくなります。

[*87] ですから，以下のプログラムを実際に Visual Basic で動作させるためには，Visual Studio 2019 および，Visual Basic の入門を済ませておく必要があります。

[*88] この篩用配列はビット配列で十分でした。ですから，ビット配列が使えれば更にメモリーを効率的に使えるのですが。

るものとすると

```
    Shared Sub Print(ByVal M As String)
        Form1.TextScreen.AppendText(M + vbNewLine)
    End Sub
```

と簡単にできます。Time$ の代替は，Date 型変数 dTime を宣言して使用します。

以上を纏めると，次が得られます。以下は，プログラム 2.1’ を 10^9 までの素数表作成するものとして Visual Basic に移植したものの主要部，Form の部分です[*89]。

プログラム **7.1**

[プログラム 2.1’ の Visual Basic への移植プログラムの Form1 の部分]

```
Public Class Form1
    Private Const MaxN As Integer = 1000000000 ' 10^9
    Private arrayNT(MaxN) As Byte
    Private dTime As Date

    Private Sub GoButton_Click(sender As Object, e As EventArgs) Handles GoButton.Click
        Dim i, p, Counter As Long
        dTime = Now
        Print(dTime.ToString)
        p = 2
        While p * p <= MaxN
            For i = p To (MaxN \ p)
                arrayNT(CInt(i * p)) = 1
            Next i
            p += 1
            While arrayNT(CInt(p)) = 1 : p += 1 : End While
        End While
        dTime = Now
        Print(dTime.ToString)
        Print("")
        Counter = 0
        For i = 2 To MaxN
            If arrayNT(CInt(i)) = 0 Then Counter += 1
        Next i
        Print("the Number of Primes less than" _
            + (MaxN).ToString + " is " + Counter.ToString + ".")
    End Sub

    Shared Sub Print(ByVal M As String)
        Form1.TextScreen.AppendText(M + vbNewLine)
    End Sub

    Private Sub ExitButton_Click(sender As Object, e As EventArgs) Handles ExitButton.Click
        Close()
    End Sub

End Class
```

[*89] 以下のプログラムで \ がありますが，これはバックスラッシュです。日本語エディターでは ¥ と表示されるものと同じコードです。

(2) 改良篩法

改良篩法 2.2' は上のプログラム 7.1 を少し修正するだけで完成します。

(3) 車輪篩法

車輪篩法 3.1 も同様に移植可能です。最大公約数関数 Gcd は Visual Basic に無いので，自前で作る必要がありますが，次のコード可能です。

```
    Private Function Gcd(a As Integer, b As Integer) As Integer
        Dim r As Integer                    ' a,b は 0 以上の整数，共に 0 ではない。
        While (b > 0)
            r = a Mod b
            a = b
            b = r
        End While
        Return a
    End Function
```

(4) リスト篩法

プログラム 4.1 は，配列変数 NextS，PrevS は 32 ビット整数 integer または 64 ビット整数 Long で宣言しなくてはなりません。そのため，MaxN の最大値は 10^8 までです。ですから実際のプログラムでは，Form1 の冒頭部分で

```
    Private Const MaxN As Integer = 100000000 '10^8
    Private NextS(MaxN) As Integer
    Private PrevS(MaxN + 1) As Integer
    Private dTime As Date
```

と宣言します。後はほぼそのままで移植可能です。

(5) 既約篩法

既約篩法では，プログラム 5.4 を移植することにします。

既約篩法は，既約な表（$\mathrm{mP_n}$ と互いに素な表）を篩うことで高速化を図るものですが，そのために表との変換関数 $\mathrm{Rp_n}$，$\mathrm{Ir_n}$ を使っています。これらを使って篩の本体部分は

```
For i0 = i To iMax
   NT(Irn(Rpn(i0) * p)) = 1
Next i0
```

と表されました。そしてこの計算 Irn(Rpn(i0) * p) が如何に高速に行えるかが鍵となります。その高速化のための方法が表 rP，iR の利用でした。それにより例えば，$\mathrm{Rp_n}$ は tbasic での例では

```
Function Rpn(i)
   Rpn = mPn * Int(i / phiPn) + rP(i mod phiPn)
End Function
```

と書かれていました。ここで，Int(i/phiPn) は，(i \ phiPn) と同じでしたが，Visual Basic には，i を phiPn で割ったときの商と余りを同時に計算する手続き Math.DivRem があります。これを使うと，

```
Function Rpn(i As Long) As Long
    q = Math.DivRem(i, phiPn, r)
    Return mPn * q + rP(CInt(r))
End Function
```

と書くことができます。この方がより高速です。Irn も同様です。しかし，一般に関数を呼ぶ処理はそれだけで一定の処理時間を要します。上の計算 Irn(Rpn(i0) * p) では，1 回の篩で 2 回関数を呼んでいます。より高速を目指すとすれば，この部分を関数を使用しないで書く方法があります。ここでは，多少プログラムが複雑になりますが，繰り返し回数の多い部分の Rpn，Irn を関数呼び出しではなく，当該部分に直接関数内のコードを次のように書くことにします。これによりかなりの高速化が得られます。

```
i = 2
p = Rpn(i)
sMaxN = Rpn(MaxN)
While p * p <= sMaxN
    i0 = sMaxN \ p
    i1 = i0 + fL(CInt(i0 Mod mPn))
    q = Math.DivRem(i1, mPn, r)
    iMax = (phiPn * q) + iR(CInt(r)) ' iMax = Irn(FloorN(sMaxN \ p))
    For i0 = i To iMax
        q = Math.DivRem(i0, phiPn, r)
        i1 = (mPn * q) + rP(CInt(r))   ' i1 = Rpn(i0)
        i1 *= p
        q = Math.DivRem(i1, mPn, r)
        i1 = (phiPn * q) + iR(CInt(r)) ' i1 = Irn(Rpn(i0)*p)
        arrayNT(CInt(i1)) = 1            '
    Next i0
    i += 1
    While arrayNT(CInt(i)) = 1 : i += 1 : End While
    q = Math.DivRem(i, phiPn, r)
    p = mPn * q + rP(CInt(r))  ' p=Rpn(i)
End While
```

既約篩の n の設定は，Form1 の冒頭部で例えば，$n = 6$ で，10^{10} 迄篩うには，次のように宣言します。

```
Private Const mPn As Integer = 30030  ' 2*3*5*7*11*13
Private Const phiPn As Integer = 5760  ' 1*2*4*6*10*12 = phi(30030)
Private Const MaxN As Integer = 1918081917 ' 10^10
```

(6) 実行結果

上のプログラムを実行した結果は以下の通りです。

	基本篩	改良篩	車輪篩(w_4)	車輪篩(w_6)	リスト篩	既約篩($n5$)	既約篩($n6$)	既約篩($n7$)
10^8	1秒	1秒	1秒	1秒	3秒	1秒未満	1秒未満	1秒未満
$5 \cdot 10^8$	5秒	3秒	4秒	4秒	–	1秒	1秒	1秒
10^9	10秒	6秒	9秒	8秒	–	2秒	2秒	3秒
$5 \cdot 10^9$	–	–	–	–	–	13秒	12秒	14秒
10^{10}	–	–	–	–	–	27秒	25秒	30秒

tbasic での実行結果と同様な傾向ですが，速度は 100 倍以上の高速です。また，既約篩法は大きな数表を作れるだけでなく，素朴な方法に比べてより高速であることが分かります。

7.2.2 C# での比較

上のプログラムをそれぞれ C# に移植して，速度を比較してみましょう。Visual Studio 上での移植なので，Visual Basic の場合と同様に C# の project で Form を構成して，プログラム本体の部分の書き直せば良いだけです。

上のプログラムで使われている構文は基本的なものばかりですから，C# の文法に従ってほぼ逐語的に書き直すことができます。ただ，配列の宣言については少し注意が必要です。Basic で dim A(10) とすると，$A(0)$ ～$A(10)$ の配列が確保されますが，C では，intA[10] とすると，確保されるのは A(0)～A(9) です。

このことから例えば，プログラム 7.1 に対応する C# のプログラムでは，

```
private static readonly int MaxN = 1000000000;// 10^9
private static byte[] arrayNT = new byte[MaxN + 1];
```

となります。

移植の雰囲気の例として，リスト篩の篩部分のプログラム一部分を次に挙げます。

プログラム **7.2**

[プログラム 4.1’の C# への移植プログラムの Form1 の一部分]

```
private const int MaxN = 100000000;
private int[] NextS= new int[MaxN+1];
private int[] PrevS = new int[MaxN + 2];
private DateTime dTime;

private void EratosthenesGo_Click(object sender, EventArgs e)
{
    int i, Counter, p;
    dTime = DateTime.Now;
    Print(dTime.ToString());
    for (i = 2; i <= MaxN; i++)
    {
        NextS[i] = i + 1;
        PrevS[i] = i - 1;
    }
    p = 2;
    dTime = DateTime.Now;
    Print(dTime.ToString());
    while (p*p<=MaxN)
    {   i = p;
        while(NextS[i]*p <= MaxN)
        {
            i = NextS[i];
        }
        do
        {   PrevS[NextS[i * p]] = PrevS[i * p];
            NextS[PrevS[i * p]] = NextS[i * p];
            i = PrevS[i];
        } while (i >= p);
        p = NextS[p];
    }
    dTime = DateTime.Now;
    Print(dTime.ToString());
    Counter = 1;
```

```
            p = 2;
            while (p <= MaxN)
            {   p = NextS[p];
                if (p < MaxN)
                {
                    Counter += 1;
                };
            };
            Print("The number of primes less than" + MaxN.ToString() + " is "
                        + Counter.ToString());
        }
    }
```

以上のプログラムを実行した結果は次の通りです。

	基本篩	改良篩	車輪篩(w_4)	車輪篩(w_6)	リスト篩	既約篩($n5$)	既約篩($n6$)	既約篩($n7$)
10^8	1秒	1秒	1秒	1秒	3秒	1秒未満	1秒未満	1秒未満
$5\cdot10^8$	5秒	3秒	5秒	4秒	–	4秒	2秒	2秒
10^9	10秒	6秒	11秒	10秒	–	4秒	3秒	3秒
$5\cdot10^9$	–	–	–	–	–	14秒	13秒	15秒
10^{10}	–	–	–	–	–	28秒	27秒	31秒

結果を見ると，Visual Basic とほぼ同じ，むしろ多少遅いものもあり，意外な感じです。

7.2.3 10^{13} 迄の素数表の作成

上の結果から，既約篩法を用いれば 10^{10} 迄の素数表は Visual Basic でも C# でも 30 秒前後で一度に作成できることが分かりました。ですから，区分法を使えばこれらの約 1000 倍で 10^{13} 迄の素数表が作成できることになります。

そこで，プログラム 6.2 を 10^{13} 迄の素数表作成用に C# に移植してみました[*90]。

プログラム 5.4 の移植と同様ですが，区分法での篩は，単独の篩に比べてメモリーを使いますから，ここでは $n=7$ ではなくて，$n=6$ の区分法を使うことにします。この場合，$n=6$ 用として，いくつか定数の調整が必要になります。

まず，篩うために $\sqrt{10^{13}}=3162277....$ 以下の素数を使います。脱出条件として，一つ加えて，$\sqrt{10^{13}}$ を超える最初の素数 3162283 まで用意します。$\sqrt{10^{13}}$ 以下の素数の個数は 227647 ですから，素数表用配列は 227648 まで宣言します。

ii を 1〜1000 まで動かしたとき，最大の MaxN は計算すると，1918081923 になります。これらの宣言部分は次のようになります。

```
        private static readonly int mPn = 30030;
        private static readonly int phiPn = 5760;

        private static readonly int MaxNN = 1918081923;// 10^10 での最大篩幅

        private static int MaxPD = 227647 + 1; //π (sqrt(10^13))+1
```

[*90] Visual Basic にも移植しましたが，10^{10} を使った部分篩は $n=6$ でもメモリー不足で動作しませんでした。また，$5\cdot10^9$ 用には可能でしたが，速度が単独版より遅くなり，Visual Basic での動作を諦めました。

```
private static int MaxSPN = 3162283;   // sqrt(10 ^ 13) を超える最初の素数

private static byte[] arrayNT = new byte[MaxNN + 1];
private static int[] rP = new int[phiPn + 1];
private static int[] iR = new int[mPn + 1];
private static int[] fL = new int[mPn + 1];
private static int[] cE = new int[mPn + 1];
private static int[] primes = new int[MaxPD + 1];
```

これらの調整を施したものを実行した結果，約 40 秒で一回の篩が可能でした。区分法ではいくつかの追加的処理が必要ですので，単独版よりも時間が掛かります。これらのことから，12 時間程度で 10^{13} の篩が実行できると予想されます。

単独版の篩では，配列変数の値は，篩が終われば，素数表を表します。ですからこれで素数表が作成されたと見ることもできます。しかし，区分法では，新たな区分を篩う際に以前の篩の結果（配列変数の値）は初期化します。ですから，区分法で篩い，全体としての素数表を得るとすれば，各区分での篩の結果を外部に保存する必要があります。PC で外部に保存するには，ファイルを使うのが標準的です。次節ではこの問題を考えてみましょう。

7.3 素数ファイル

篩を行った結果は，プログラムのメモリー上にある場合は，プログラムを終了したり，PC の電源を切ると消えてしまします。ですから，篩で作成された素数表を保存しておくには，素数表をファイルに保存する必要があります。ここでは，素数をファイルに保存する方法を考えてみます。

7.3.1 素朴な方法

最も自然に考えられる方法は，テキストファイルにそのまま素数を記述する方法です。例えば，10 以下の素数表として

```
2
3
5
7
```

と言ったファイルを利用するものです。これは素数の個数だけの行数を持つテキストファイルです。この方法は内容がすぐ分かり，使い易いものです[*91]。

この方法で 10^8 までの素数表ファイルを作成すると，約 54 メガバイトのファイルになります[*92]。

10^{13} 迄の素数の個数についてはかなり詳しく知られていて[*93]，その結果を使うと，この方法で，10^{13} までの素数表のファイルを作ると，約 4796 ギガバイトになることが分かります。現在利用できるハードディスクでこれ以上の容量を持つものもありますから，この方法で保存することも不可能ではありません。しかし，大容量のハードディスクがこのファイルだけで一杯に成るほどの大きさです。そして勿論このような大きさのファイルを扱うことはできませんから，いくつかに分割して保存する必要があります。

[*91] この方法で，Windows 上のテキストファイルとして保存すると 1 行が (数字の個数 +2) バイト必要です。また Linux や Mac の場合は (数字の個数 +1) バイトです。

[*92] 勿論このファイルを圧縮すれば，約 13 メガバイトと更に小さくなります。

[*93] もっと大きな数についても知られています。

7.3.2 素数間の差を格納する

3 以上の素数はすべて奇数ですから，k 番目の素数を p_k と表すとき，$k \geqq 2$ なら，$p_{k+1} - p_k \geqq 2$ となりまが，この差 $p_{k+1} - p_k$ は余り大きくなりません。ですから，この差を格納すれば，素数そのものを格納するよりファイルサイズが小さくなります。例えば，

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31$$

の代わりに，それらの差

$$2, 1, 2, 2, 4, 2, 4, 2, 4, 6, 2$$

を格納します。この素数間の差を一般的に簡単に求めることは難しいですが，実際に計算機で計算してみるのは，素数表があれば簡単です。この方法で 10^8 迄の素数表ファイルを作成すると，約 20 メガバイトをになります[*94]。素朴な方法よりも小さくなることが分かります。実は，10^{13} 以下の素数間の差は高々 3 桁の数であることが知られています。ですから，多くとも 1600 ギガバイトで収まることが分かります。そして，後述する【表 1：10^{13} 以下の素数間隔】を使えば，実際に約 1300 ギガバイトのファイルなることが分かります。この方法を使うと素朴な方法よりも大分小さくなりますが，その分扱いは多少面倒です。つまり，ファイルの中身を単純に見ただけでは，ある数が素数であるかは分かりません。

7.3.3 篩の配列をファイルに保存する方法

篩を行ったとき，配列を使いました。そしてその配列は実際には 0, 1 だけ格納すれば十分でした。そこで，この配列と同じものをファイルに格納すれば素数表を保存したことになります。例えば，10 までの素数表 `2 3 5 7` を格納する替わりに，位置を数，0 を素数を表すとして，`1001010111` で表します。

この方法だと，上限バイト数必要ですから，例えば，10^8 までの素数表をこの方法でファイルとして保存すると，約 95 メガバイト必要です[*95]。10^{13} までの素数表だと，9314 ギガバイトになります。

また，バイトではなくビットで表すことにすれば，この 1/8 になります。ですから，10^{13} までの素数表をビット列ファイルとして保存すると，約 1165 ギガバイトになります。更に，既約篩法の技法を使って，例えば，$n = 6$ とすると，約 1/5 になります。ですから，約 230 ギガバイト程度のファイルで 10^{13} までの素数表が格納可能です。

既約篩法の技法を使って，ビット配列として保存すると，素朴な方法に比べてかなり効率的に保存ができることが分かります。

しかし，この方法ではファイル化された素数表から，素数を読み取るための専用のプログラムが必要になります。そして，それらの素数表を利用するのに，それなりの計算処理が必要で，時間も掛かることになります。

このように効率的に保存すればするほど使いづらくなるという面があります。

[*94] これを圧縮すると約 5 メガバイトになります。

[*95] 勿論，2 を除けば素数はすべて奇数ですから奇数の素数表を格納するとすれば，ファイルの大きさは約半分になります。

7.4 数表

10^{13} 以下の素数表を作成する篩は一定の時間を掛ければ可能でした。しかし，前節で見たように，10^{13} 以下の素数表はどのようなファイル形式で保存しても膨大です。また仮に保存したとしても，それが正しいかどうかの検証や，それを何に利用するのかの問題もあります。そこで，篩を行いますが，その結果の表はファイルに保存しないで，篩を行う途中で中間的に得られる素数表を使って２種類の計算を行い，その結果を示すことで，篩を行ったこのとの検証とすることにしました*96。

(1) 素数間隔

連続する２素数の間隔がどのくらいあるかは，素数の性質で興味の湧く問題です。これについてはいくつかの予想も成されています。これに関連する最もよく知られた簡単な性質は

- 任意の整数 $n > 0$ に対して，$p_{k+1} - p_k > n$ となる連続する素数 p_k，p_{k+1} が存在する。

です。ここでは，単純に 10^{13} 以下の素数について，$p_{k+1} - p_k$ を計算し，その個数を数えることにします。

そのために，次のように構造体を用意し, その配列を使います*97。

```
private struct PrimDis
{
    public long dCount;
    public long firstPrime;
}

private PrimDis[] PrimeDistance = new PrimDis[800];
```

各区分ごとに，PrimeDistance を計算し，プログラム終了時にその結果を，その区分での最大素数とともにテキストファイルに保存します。次に計算を開始する際，そのテキストファイルの内容を PrimeDistance に読み込み，計算を始めます。ii = 1000 まで計算を行った結果が 10^{13} 迄の素数間隔の【表 1】になります。

素数間隔	個数	最小素数p_{k+1}	素数間隔	個数	最小素数p_{k+1}
1	1	3	100	584268567	396833
2	15834664871	5	102	828397685	1444411
4	15834656003	11	104	394492758	1388587

3 以上の素数間隔は偶数ですから，3 以上の奇数間隔の部分は表示していません。最小素数はその素数間隔が表れる最初の素数です。例えば素数間隔 100 に対する最小素数 396833 はその直前の素数が 396733 であることを意味します。

(2) 素数の個数

n 以下の素数の個数を $\pi(n)$ と表します。例えば，$\pi(2) = 1$，$\pi(6) = 3$，$\pi(100) = 25$ です。素数の個数は素数分布との関係で詳しく調べられています。数学的な関係だけでなく，具体的な $\pi(n)$ の値についてもかなり大きな数迄計算されています*98。$\pi(n)$ は素数表があればその個数を数えるだけですから簡単に計算できます。

ここでは，各区間ごと篩の結果から素数を数え，それを集計します。ii = 1 から ii = 1000 までの各区間ごとに集計していけば 10^{10} 間隔での $\pi(n)$ の【表 2】になります。

---

*96 以下の表は既に知られている結果（例えば，Hans Riesel, Prime Numbers and Computer Methods for Factorization, Birkhäuser,1994）と整合的ですので，正しく計算されたと思われます。

*97 tbasic で同じことを行うとすれば，２種の配列変数を使います。計算してみると，最大の素数間隔は 674 であることが分かりますが，ここでは少し多く 799 まで用意しています。

*98 例えば，$\pi(10^{25}) = 176,846,309,399,143,769,411,680$ だそうです。勿論，この値は素数表で数えたものではありません。

【表 1：10^{13} 以下の素数間隔】

素数間隔	個数	最小素数p_{k+1}	素数間隔	個数	最小素数p_{k+1}
1	1	3	100	584268567	396833
2	15834664871	5	102	828397685	1444411
4	15834656003	11	104	394492758	1388587
6	29289695650	29	106	351669276	1098953
8	13515180171	97	108	639388319	2238931
10	17663498098	149	110	413181898	1468387
12	23931289984	211	112	327700129	370373
14	13597962107	127	114	523318493	492227
16	10576414202	1847	116	234398970	5845309
18	19878698732	541	118	230302500	1349651
20	11638466683	907	120	524351328	1895479
22	9888091790	1151	122	177333152	3117421
24	15861755259	1693	124	181174317	6752747
26	7902711871	2503	126	368513409	1671907
28	8643973155	2999	128	138379585	3851587
30	16694681959	4327	130	192891055	5518817
32	5582122930	5623	132	261418550	1357333
34	5900013825	1361	134	113357312	6958801
36	10109375268	9587	136	108328192	6371537
38	4957098874	30631	138	205176276	3826157
40	5994718578	19373	140	136684236	7621399
42	9484975460	16183	142	83101585	10343903
44	4043811134	15727	144	150160276	11981587
46	3604148124	81509	146	67926519	6034393
48	6443528078	28277	148	72284291	2010881
50	3862450797	31957	150	160536036	13626407
52	3003793520	19661	152	55183966	8421403
54	5167085638	35671	154	68766000	4652507
56	2816219685	82129	156	100793223	17983873
58	2433024862	44351	158	43733848	49269739
60	5401237011	43391	160	53520200	33803849
62	1820731588	34123	162	73049507	39175379
64	1827362938	89753	164	36119026	20285263
66	3579108813	162209	166	32520530	83751287
68	1546597364	134581	168	71964302	37305881
70	2247338579	173429	170	37065390	27915907
72	2452546905	31469	172	25212078	38394299
74	1226908425	404671	174	47864622	52721287
76	1145793024	212777	176	23221579	38089453
78	2247392410	188107	178	21211627	39390167
80	1222788009	542683	180	48692080	17051887
82	893260674	265703	182	20381260	36271783
84	1925411535	461801	184	16829486	79167917
86	735150149	156007	186	29357105	147684323
88	772368605	544367	188	12832980	134066017
90	1654272409	404941	190	18141708	142414859
92	587201625	927961	192	22231553	123454883
94	548095541	1101071	194	10484617	166726561
96	1000057637	360749	196	11680741	70396589
98	548858040	604171	198	20086507	46006967

素数間幅	個数	最小素数p_{k+1}	素数間幅	個数	最小素数p_{k+1}
200	10694134	378044179	300	397747	4758959041
202	7943753	107534789	302	133657	6675573799
204	15210222	112099021	304	135760	2433630413
206	6496216	232424029	306	254326	3917587543
208	7044631	192984059	308	141114	5490459409
210	17354386	20831533	310	143495	4024713971
212	4700828	215949619	312	196640	6570018659
214	4635033	253878617	314	83540	8948419063
216	8463973	202551883	316	81852	12109172609
218	3939160	327966319	318	152774	4373000039
220	5535299	47326913	320	86576	2300942869
222	6866198	122164969	322	80649	7961074763
224	3844816	409866547	324	115714	10958688203
226	2888868	519653597	326	53924	5837935699
228	5731984	895858267	328	54217	13086861509
230	3326329	607010323	330	134482	6291356339
232	2424688	525436721	332	39948	5893180453
234	4579706	189695893	334	39195	30827138843
236	1938920	216668839	336	83414	3842611109
238	2364904	673919381	338	34344	22076314651
240	4342176	391995671	340	42618	8605261787
242	1616463	367876771	342	56678	12010745911
244	1435076	693103883	344	26302	19724087611
246	2678323	555142307	346	23341	11291402183
248	1245699	191913031	348	46304	17002876991
250	1527319	387096383	350	31343	16808773627
252	2412770	630045389	352	20920	30750893153
254	946839	1202442343	354	34463	4302407713
256	861641	1872852203	356	15308	24355072873
258	1685172	1316355581	358	15571	16792321697
260	1073966	944193067	360	35915	20068818557
262	705890	1649329259	362	12004	35877724963
264	1422962	2357882257	364	14875	25425617681
266	720042	1438780087	366	21208	20108776463
268	577334	1579307057	368	10354	51430518781
270	1315892	1391048317	370	12422	59942358941
272	440944	1851255463	372	17124	20404138151
274	453445	1282463543	374	9131	23064762037
276	826092	649580447	376	7502	16161670163
278	352338	4260928879	378	15960	38116958197
280	562010	1855047443	380	7942	23323809121
282	616640	436273291	382	5525	10726905041
284	290572	1667186743	384	10356	20678048681
286	321516	2842739597	386	4707	35238645973
288	495479	1294268779	388	4766	156798792611
290	308125	1948819423	390	11576	53241806041
292	217409	1453168433	392	4166	117215204923
294	460605	5692630483	394	3524	22367085353
296	176788	5260030807	396	6865	50806026269
298	168438	8650524881	398	2812	40267027987

素数間幅	個数	最小素数p_{k+1}	素数間幅	個数	最小素数p_{k+1}
400	3733	47203303559	500	67	303371455741
402	5013	44293346579	502	38	1258535917103
404	2287	144999022447	504	103	747431049707
406	2702	49306638713	506	48	1339347751213
408	4134	134664608797	508	36	1841086484999
410	2440	98276144503	510	80	2209016910641
412	1774	124221464531	512	24	1999066711903
414	3290	49914935591	514	24	304599509051
416	1457	121972158853	516	55	416608696337
418	1626	129300695021	518	29	2296497058651
420	3724	82490815543	520	30	2336167262969
422	1059	280974865783	522	32	1214820696223
424	994	264495345683	524	20	2256065636563
426	1836	180265084829	526	17	1620505682897
428	890	219950168839	528	28	1529741785667
430	1061	250964194601	530	24	2205492372901
432	1494	87241771051	532	12	461690510543
434	880	127084570357	534	20	614487454057
436	623	367459060307	536	10	5371284218299
438	1192	101328529879	538	10	2122536905849
440	752	141846300241	540	29	738832928467
442	635	417470555129	542	9	2707053888193
444	983	36172730507	544	11	2652427556183
446	392	190418076649	546	17	2164206785267
448	514	402872475191	548	10	3380058341827
450	990	63816175897	550	7	2496646209821
452	277	466855187923	552	11	2210401547153
454	321	202530831617	554	4	3621153039853
456	623	25056082543	556	3	4338624362729
458	245	304040251927	558	7	5263973983381
460	343	131956236023	560	6	4260199366933
462	566	400729567543	562	4	2081209441841
464	208	42652618807	564	10	1480064231717
466	186	565855696097	566	3	4897642179763
468	383	127976335139	568	2	6010330572899
470	221	681753256603	570	5	4442109925787
472	136	865244710079	572	4	5441175347539
474	296	182226896713	574	3	3108794067653
476	170	725978934823	576	4	8817792099037
478	124	367766548049	578	2	7552870121299
480	260	482423534377	580	1	9383081341121
482	78	1051602787663	582	2	1346294311331
484	89	767644375301	584	2	6993007248823
486	159	241160624629	586	4	6364466317163
488	82	1275363152587	588	3	1408695494197
490	121	297501076289	590	0	–
492	126	910361181181	592	1	3410069454689
494	62	804541404913	594	4	5499789520457
496	59	880318999403	596	0	–
498	134	428315807321	598	2	5614481774159

素数間幅	個数	最小素数p_{k+1}	素数間幅	個数	最小素数p_{k+1}
600	1	4872634110667	640	0	–
602	1	1968188557063	642	0	–
604	2	5439564949187	644	0	–
606	0	–	646	0	–
608	0	–	648	1	9787731508409
610	2	9105981382787	650	2	5120731250857
612	0	–	652	1	2614941711251
614	0	–	654	0	–
616	1	8095224518267	656	0	–
618	1	4165633395767	658	0	–
620	1	9344093036281	660	0	–
622	0	–	662	0	–
624	0	–	664	0	–
626	0	–	666	0	–
628	0	–	668	0	–
630	0	–	670	0	–
632	0	–	672	0	–
634	0	–	674	1	7177162612387
636	1	9483480842389	676	0	–
638	0	–	678	0	–

【表 2：10^{13} 以下の素数の個数】

n	$\pi(n)$	n	$\pi(n)$	n	$\pi(n)$
$1 \cdot 10^{10}$	455052511	$51 \cdot 10^{10}$	19679230267	$101 \cdot 10^{10}$	37969752226
$2 \cdot 10^{10}$	882206716	$52 \cdot 10^{10}$	20050047970	$102 \cdot 10^{10}$	38331462353
$3 \cdot 10^{10}$	1300005926	$53 \cdot 10^{10}$	20420605272	$103 \cdot 10^{10}$	38693048890
$4 \cdot 10^{10}$	1711955433	$54 \cdot 10^{10}$	20790904461	$104 \cdot 10^{10}$	39054508955
$5 \cdot 10^{10}$	2119654578	$55 \cdot 10^{10}$	21160938999	$105 \cdot 10^{10}$	39415850926
$6 \cdot 10^{10}$	2524038155	$56 \cdot 10^{10}$	21530734955	$106 \cdot 10^{10}$	39777065237
$7 \cdot 10^{10}$	2925699539	$57 \cdot 10^{10}$	21900279704	$107 \cdot 10^{10}$	40138161844
$8 \cdot 10^{10}$	3325059246	$58 \cdot 10^{10}$	22269587413	$108 \cdot 10^{10}$	40499125125
$9 \cdot 10^{10}$	3722428991	$59 \cdot 10^{10}$	22638655971	$109 \cdot 10^{10}$	40859978323
$10 \cdot 10^{10}$	4118054813	$60 \cdot 10^{10}$	23007501786	$110 \cdot 10^{10}$	41220703418
$11 \cdot 10^{10}$	4512105232	$61 \cdot 10^{10}$	23376112935	$111 \cdot 10^{10}$	41581301356
$12 \cdot 10^{10}$	4904759399	$62 \cdot 10^{10}$	23744505064	$112 \cdot 10^{10}$	41941795312
$13 \cdot 10^{10}$	5296138250	$63 \cdot 10^{10}$	24112681510	$113 \cdot 10^{10}$	42302174835
$14 \cdot 10^{10}$	5686326158	$64 \cdot 10^{10}$	24480636752	$114 \cdot 10^{10}$	42662432368
$15 \cdot 10^{10}$	6075437956	$65 \cdot 10^{10}$	24848387400	$115 \cdot 10^{10}$	43022589332
$16 \cdot 10^{10}$	6463533937	$66 \cdot 10^{10}$	25215924549	$116 \cdot 10^{10}$	43382637099
$17 \cdot 10^{10}$	6850690633	$67 \cdot 10^{10}$	25583264393	$117 \cdot 10^{10}$	43742561726
$18 \cdot 10^{10}$	7236978160	$68 \cdot 10^{10}$	25950403691	$118 \cdot 10^{10}$	44102360955
$19 \cdot 10^{10}$	7622425159	$69 \cdot 10^{10}$	26317331556	$119 \cdot 10^{10}$	44462058956
$20 \cdot 10^{10}$	8007105059	$70 \cdot 10^{10}$	26684074310	$120 \cdot 10^{10}$	44821651132
$21 \cdot 10^{10}$	8391034591	$71 \cdot 10^{10}$	27050626945	$121 \cdot 10^{10}$	45181142965
$22 \cdot 10^{10}$	8774265364	$72 \cdot 10^{10}$	27416974273	$122 \cdot 10^{10}$	45540525006
$23 \cdot 10^{10}$	9156827831	$73 \cdot 10^{10}$	27783156146	$123 \cdot 10^{10}$	45899794955
$24 \cdot 10^{10}$	9538769484	$74 \cdot 10^{10}$	28149147782	$124 \cdot 10^{10}$	46258961316
$25 \cdot 10^{10}$	9920079604	$75 \cdot 10^{10}$	28514968374	$125 \cdot 10^{10}$	46618036665
$26 \cdot 10^{10}$	10300824253	$76 \cdot 10^{10}$	28880598062	$126 \cdot 10^{10}$	46976989669
$27 \cdot 10^{10}$	10681008150	$77 \cdot 10^{10}$	29246057094	$127 \cdot 10^{10}$	47335846514
$28 \cdot 10^{10}$	11060658751	$78 \cdot 10^{10}$	29611342201	$128 \cdot 10^{10}$	47694607668
$29 \cdot 10^{10}$	11439794944	$79 \cdot 10^{10}$	29976439588	$129 \cdot 10^{10}$	48053260272
$30 \cdot 10^{10}$	11818439135	$80 \cdot 10^{10}$	30341383527	$130 \cdot 10^{10}$	48411812843
$31 \cdot 10^{10}$	12196603558	$81 \cdot 10^{10}$	30706162642	$131 \cdot 10^{10}$	48770267763
$32 \cdot 10^{10}$	12574308380	$82 \cdot 10^{10}$	31070780007	$132 \cdot 10^{10}$	49128623763
$33 \cdot 10^{10}$	12951576227	$83 \cdot 10^{10}$	31435225918	$133 \cdot 10^{10}$	49486888038
$34 \cdot 10^{10}$	13328400814	$84 \cdot 10^{10}$	31799512115	$134 \cdot 10^{10}$	49845063717
$35 \cdot 10^{10}$	13704806310	$85 \cdot 10^{10}$	32163645177	$135 \cdot 10^{10}$	50203126780
$36 \cdot 10^{10}$	14080812200	$86 \cdot 10^{10}$	32527622185	$136 \cdot 10^{10}$	50561098373
$37 \cdot 10^{10}$	14456422318	$87 \cdot 10^{10}$	32891437535	$137 \cdot 10^{10}$	50918991157
$38 \cdot 10^{10}$	14831655048	$88 \cdot 10^{10}$	33255109885	$138 \cdot 10^{10}$	51276792563
$39 \cdot 10^{10}$	15206510873	$89 \cdot 10^{10}$	33618624982	$139 \cdot 10^{10}$	51634478530
$40 \cdot 10^{10}$	15581005657	$90 \cdot 10^{10}$	33981987586	$140 \cdot 10^{10}$	51992079337
$41 \cdot 10^{10}$	15955163563	$91 \cdot 10^{10}$	34345216400	$141 \cdot 10^{10}$	52349588740
$42 \cdot 10^{10}$	16328971825	$92 \cdot 10^{10}$	34708300245	$142 \cdot 10^{10}$	52707004423
$43 \cdot 10^{10}$	16702450413	$93 \cdot 10^{10}$	35071228608	$143 \cdot 10^{10}$	53064352681
$44 \cdot 10^{10}$	17075613246	$94 \cdot 10^{10}$	35434025822	$144 \cdot 10^{10}$	53421593843
$45 \cdot 10^{10}$	17448448326	$95 \cdot 10^{10}$	35796689285	$145 \cdot 10^{10}$	53778750216
$46 \cdot 10^{10}$	17820976186	$96 \cdot 10^{10}$	36159205628	$146 \cdot 10^{10}$	54135815480
$47 \cdot 10^{10}$	18193204773	$97 \cdot 10^{10}$	36521577782	$147 \cdot 10^{10}$	54492797320
$48 \cdot 10^{10}$	18565127861	$98 \cdot 10^{10}$	36883821436	$148 \cdot 10^{10}$	54849675648
$49 \cdot 10^{10}$	18936775942	$99 \cdot 10^{10}$	37245934597	$149 \cdot 10^{10}$	55206486339
$50 \cdot 10^{10}$	19308136142	$100 \cdot 10^{10}$	37607912018	$150 \cdot 10^{10}$	55563209929

n	$\pi(n)$	n	$\pi(n)$	n	$\pi(n)$
$151 \cdot 10^{10}$	55919845463	$201 \cdot 10^{10}$	73654930802	$251 \cdot 10^{10}$	91233185143
$152 \cdot 10^{10}$	56276395420	$202 \cdot 10^{10}$	74007893510	$252 \cdot 10^{10}$	91583416850
$153 \cdot 10^{10}$	56632865046	$203 \cdot 10^{10}$	74360786856	$253 \cdot 10^{10}$	91933588482
$154 \cdot 10^{10}$	56989244925	$204 \cdot 10^{10}$	74713637557	$254 \cdot 10^{10}$	92283719328
$155 \cdot 10^{10}$	57345546894	$205 \cdot 10^{10}$	75066420528	$255 \cdot 10^{10}$	92633796033
$156 \cdot 10^{10}$	57701769415	$206 \cdot 10^{10}$	75419139390	$256 \cdot 10^{10}$	92983814800
$157 \cdot 10^{10}$	58057915844	$207 \cdot 10^{10}$	75771792820	$257 \cdot 10^{10}$	93333791866
$158 \cdot 10^{10}$	58413963735	$208 \cdot 10^{10}$	76124388117	$258 \cdot 10^{10}$	93683726862
$159 \cdot 10^{10}$	58769941497	$209 \cdot 10^{10}$	76476925285	$259 \cdot 10^{10}$	94033610466
$160 \cdot 10^{10}$	59125832286	$210 \cdot 10^{10}$	76829404177	$260 \cdot 10^{10}$	94383447111
$161 \cdot 10^{10}$	59481643316	$211 \cdot 10^{10}$	77181815097	$261 \cdot 10^{10}$	94733234307
$162 \cdot 10^{10}$	59837389135	$212 \cdot 10^{10}$	77534161269	$262 \cdot 10^{10}$	95082981285
$163 \cdot 10^{10}$	60193046703	$213 \cdot 10^{10}$	77886473487	$263 \cdot 10^{10}$	95432682264
$164 \cdot 10^{10}$	60548629848	$214 \cdot 10^{10}$	78238723000	$264 \cdot 10^{10}$	95782333546
$165 \cdot 10^{10}$	60904147299	$215 \cdot 10^{10}$	78590899650	$265 \cdot 10^{10}$	96131929359
$166 \cdot 10^{10}$	61259567840	$216 \cdot 10^{10}$	78943022386	$266 \cdot 10^{10}$	96481497654
$167 \cdot 10^{10}$	61614925562	$217 \cdot 10^{10}$	79295090325	$267 \cdot 10^{10}$	96831010271
$168 \cdot 10^{10}$	61970198559	$218 \cdot 10^{10}$	79647097902	$268 \cdot 10^{10}$	97180473470
$169 \cdot 10^{10}$	62325413267	$219 \cdot 10^{10}$	79999050261	$269 \cdot 10^{10}$	97529906799
$170 \cdot 10^{10}$	62680547806	$220 \cdot 10^{10}$	80350940550	$270 \cdot 10^{10}$	97879280182
$171 \cdot 10^{10}$	63035605333	$221 \cdot 10^{10}$	80702787335	$271 \cdot 10^{10}$	98228608654
$172 \cdot 10^{10}$	63390584886	$222 \cdot 10^{10}$	81054571394	$272 \cdot 10^{10}$	98577891236
$173 \cdot 10^{10}$	63745500860	$223 \cdot 10^{10}$	81406294053	$273 \cdot 10^{10}$	98927134923
$174 \cdot 10^{10}$	64100319327	$224 \cdot 10^{10}$	81757974597	$274 \cdot 10^{10}$	99276335445
$175 \cdot 10^{10}$	64455078297	$225 \cdot 10^{10}$	82109594348	$275 \cdot 10^{10}$	99625495641
$176 \cdot 10^{10}$	64809777588	$226 \cdot 10^{10}$	82461169136	$276 \cdot 10^{10}$	99974609219
$177 \cdot 10^{10}$	65164413081	$227 \cdot 10^{10}$	82812682509	$277 \cdot 10^{10}$	100323670334
$178 \cdot 10^{10}$	65518955602	$228 \cdot 10^{10}$	83164140956	$278 \cdot 10^{10}$	100672696168
$179 \cdot 10^{10}$	65873437926	$229 \cdot 10^{10}$	83515547322	$279 \cdot 10^{10}$	101021670066
$180 \cdot 10^{10}$	66227858344	$230 \cdot 10^{10}$	83866890515	$280 \cdot 10^{10}$	101370585277
$181 \cdot 10^{10}$	66582197785	$231 \cdot 10^{10}$	84218182836	$281 \cdot 10^{10}$	101719482617
$182 \cdot 10^{10}$	66936471190	$232 \cdot 10^{10}$	84569414034	$282 \cdot 10^{10}$	102068329654
$183 \cdot 10^{10}$	67290674322	$233 \cdot 10^{10}$	84920599612	$283 \cdot 10^{10}$	102417120021
$184 \cdot 10^{10}$	67644807037	$234 \cdot 10^{10}$	85271734199	$284 \cdot 10^{10}$	102765885442
$185 \cdot 10^{10}$	67998866243	$235 \cdot 10^{10}$	85622817647	$285 \cdot 10^{10}$	103114592809
$186 \cdot 10^{10}$	68352869011	$236 \cdot 10^{10}$	85973848689	$286 \cdot 10^{10}$	103463256745
$187 \cdot 10^{10}$	68706788124	$237 \cdot 10^{10}$	86324829661	$287 \cdot 10^{10}$	103811904862
$188 \cdot 10^{10}$	69060652171	$238 \cdot 10^{10}$	86675756606	$288 \cdot 10^{10}$	104160497928
$189 \cdot 10^{10}$	69414439786	$239 \cdot 10^{10}$	87026636665	$289 \cdot 10^{10}$	104509047149
$190 \cdot 10^{10}$	69768165361	$240 \cdot 10^{10}$	87377470252	$290 \cdot 10^{10}$	104857545428
$191 \cdot 10^{10}$	70121829126	$241 \cdot 10^{10}$	87728237041	$291 \cdot 10^{10}$	105206007798
$192 \cdot 10^{10}$	70475414424	$242 \cdot 10^{10}$	88078962247	$292 \cdot 10^{10}$	105554431933
$193 \cdot 10^{10}$	70828947908	$243 \cdot 10^{10}$	88429636866	$293 \cdot 10^{10}$	105902808805
$194 \cdot 10^{10}$	71182431689	$244 \cdot 10^{10}$	88780248235	$294 \cdot 10^{10}$	106251148907
$195 \cdot 10^{10}$	71535816028	$245 \cdot 10^{10}$	89130813875	$295 \cdot 10^{10}$	106599450598
$196 \cdot 10^{10}$	71889165530	$246 \cdot 10^{10}$	89481338559	$296 \cdot 10^{10}$	106947711341
$197 \cdot 10^{10}$	72242443044	$247 \cdot 10^{10}$	89831797098	$297 \cdot 10^{10}$	107295913540
$198 \cdot 10^{10}$	72595648625	$248 \cdot 10^{10}$	90182223394	$298 \cdot 10^{10}$	107644080308
$199 \cdot 10^{10}$	72948808364	$249 \cdot 10^{10}$	90532598552	$299 \cdot 10^{10}$	107992212935
$200 \cdot 10^{10}$	73301896139	$250 \cdot 10^{10}$	90882915772	$300 \cdot 10^{10}$	108340298703

n	$\pi(n)$	n	$\pi(n)$	n	$\pi(n)$
$301 \cdot 10^{10}$	108688353147	$351 \cdot 10^{10}$	126042442622	$401 \cdot 10^{10}$	143310800317
$302 \cdot 10^{10}$	109036364560	$352 \cdot 10^{10}$	126388600969	$402 \cdot 10^{10}$	143655379879
$303 \cdot 10^{10}$	109384344147	$353 \cdot 10^{10}$	126734729044	$403 \cdot 10^{10}$	143999909382
$304 \cdot 10^{10}$	109732286421	$354 \cdot 10^{10}$	127080820055	$404 \cdot 10^{10}$	144344435245
$305 \cdot 10^{10}$	110080165264	$355 \cdot 10^{10}$	127426883971	$405 \cdot 10^{10}$	144688911620
$306 \cdot 10^{10}$	110428020123	$356 \cdot 10^{10}$	127772910236	$406 \cdot 10^{10}$	145033385463
$307 \cdot 10^{10}$	110775839690	$357 \cdot 10^{10}$	128118910235	$407 \cdot 10^{10}$	145377804914
$308 \cdot 10^{10}$	111123608502	$358 \cdot 10^{10}$	128464871458	$408 \cdot 10^{10}$	145722197706
$309 \cdot 10^{10}$	111471349994	$359 \cdot 10^{10}$	128810796579	$409 \cdot 10^{10}$	146066567031
$310 \cdot 10^{10}$	111819048275	$360 \cdot 10^{10}$	129156678096	$410 \cdot 10^{10}$	146410910264
$311 \cdot 10^{10}$	112166709129	$361 \cdot 10^{10}$	129502538420	$411 \cdot 10^{10}$	146755232233
$312 \cdot 10^{10}$	112514314295	$362 \cdot 10^{10}$	129848364656	$412 \cdot 10^{10}$	147099527215
$313 \cdot 10^{10}$	112861886026	$363 \cdot 10^{10}$	130194158209	$413 \cdot 10^{10}$	147443787179
$314 \cdot 10^{10}$	113209423537	$364 \cdot 10^{10}$	130539922425	$414 \cdot 10^{10}$	147788012165
$315 \cdot 10^{10}$	113556935119	$365 \cdot 10^{10}$	130885655469	$415 \cdot 10^{10}$	148132213361
$316 \cdot 10^{10}$	113904403412	$366 \cdot 10^{10}$	131231345415	$416 \cdot 10^{10}$	148476393288
$317 \cdot 10^{10}$	114251834077	$367 \cdot 10^{10}$	131577007930	$417 \cdot 10^{10}$	148820536400
$318 \cdot 10^{10}$	114599223161	$368 \cdot 10^{10}$	131922639936	$418 \cdot 10^{10}$	149164641029
$319 \cdot 10^{10}$	114946587784	$369 \cdot 10^{10}$	132268228639	$419 \cdot 10^{10}$	149508722589
$320 \cdot 10^{10}$	115293888086	$370 \cdot 10^{10}$	132613795649	$420 \cdot 10^{10}$	149852773964
$321 \cdot 10^{10}$	115641153828	$371 \cdot 10^{10}$	132959323593	$421 \cdot 10^{10}$	150196795325
$322 \cdot 10^{10}$	115988395780	$372 \cdot 10^{10}$	133304824512	$422 \cdot 10^{10}$	150540803270
$323 \cdot 10^{10}$	116335595747	$373 \cdot 10^{10}$	133650300656	$423 \cdot 10^{10}$	150884784112
$324 \cdot 10^{10}$	116682753677	$374 \cdot 10^{10}$	133995741967	$424 \cdot 10^{10}$	151228739535
$325 \cdot 10^{10}$	117029875191	$375 \cdot 10^{10}$	134341145407	$425 \cdot 10^{10}$	151572659855
$326 \cdot 10^{10}$	117376967604	$376 \cdot 10^{10}$	134686518280	$426 \cdot 10^{10}$	151916548352
$327 \cdot 10^{10}$	117724021032	$377 \cdot 10^{10}$	135031853613	$427 \cdot 10^{10}$	152260403281
$328 \cdot 10^{10}$	118071027178	$378 \cdot 10^{10}$	135377158742	$428 \cdot 10^{10}$	152604240809
$329 \cdot 10^{10}$	118417998867	$379 \cdot 10^{10}$	135722427456	$429 \cdot 10^{10}$	152948047306
$330 \cdot 10^{10}$	118764942011	$380 \cdot 10^{10}$	136067688083	$430 \cdot 10^{10}$	153291810623
$331 \cdot 10^{10}$	119111842094	$381 \cdot 10^{10}$	136412919140	$431 \cdot 10^{10}$	153635568537
$332 \cdot 10^{10}$	119458710298	$382 \cdot 10^{10}$	136758107110	$432 \cdot 10^{10}$	153979289835
$333 \cdot 10^{10}$	119805544239	$383 \cdot 10^{10}$	137103260109	$433 \cdot 10^{10}$	154322989697
$334 \cdot 10^{10}$	120152348322	$384 \cdot 10^{10}$	137448378684	$434 \cdot 10^{10}$	154666660313
$335 \cdot 10^{10}$	120499088838	$385 \cdot 10^{10}$	137793473050	$435 \cdot 10^{10}$	155010287841
$336 \cdot 10^{10}$	120845821659	$386 \cdot 10^{10}$	138138536751	$436 \cdot 10^{10}$	155353910687
$337 \cdot 10^{10}$	121192507439	$387 \cdot 10^{10}$	138483561770	$437 \cdot 10^{10}$	155697508857
$338 \cdot 10^{10}$	121539165430	$388 \cdot 10^{10}$	138828555205	$438 \cdot 10^{10}$	156041071716
$339 \cdot 10^{10}$	121885775866	$389 \cdot 10^{10}$	139173517991	$439 \cdot 10^{10}$	156384615863
$340 \cdot 10^{10}$	122232349643	$390 \cdot 10^{10}$	139518461664	$440 \cdot 10^{10}$	156728136258
$341 \cdot 10^{10}$	122578888972	$391 \cdot 10^{10}$	139863368126	$441 \cdot 10^{10}$	157071622045
$342 \cdot 10^{10}$	122925400738	$392 \cdot 10^{10}$	140208253771	$442 \cdot 10^{10}$	157415066944
$343 \cdot 10^{10}$	123271877167	$393 \cdot 10^{10}$	140553102861	$443 \cdot 10^{10}$	157758495665
$344 \cdot 10^{10}$	123618330284	$394 \cdot 10^{10}$	140897921188	$444 \cdot 10^{10}$	158101901298
$345 \cdot 10^{10}$	123964739787	$395 \cdot 10^{10}$	141242717075	$445 \cdot 10^{10}$	158445276571
$346 \cdot 10^{10}$	124311096904	$396 \cdot 10^{10}$	141587481679	$446 \cdot 10^{10}$	158788626349
$347 \cdot 10^{10}$	124657427395	$397 \cdot 10^{10}$	141932197305	$447 \cdot 10^{10}$	159131946855
$348 \cdot 10^{10}$	125003735054	$398 \cdot 10^{10}$	142276896829	$448 \cdot 10^{10}$	159475249458
$349 \cdot 10^{10}$	125350005352	$399 \cdot 10^{10}$	142621569177	$449 \cdot 10^{10}$	159818514828
$350 \cdot 10^{10}$	125696244675	$400 \cdot 10^{10}$	142966208126	$450 \cdot 10^{10}$	160161747327

n	$\pi(n)$	n	$\pi(n)$	n	$\pi(n)$
$451 \cdot 10^{10}$	160504950470	$501 \cdot 10^{10}$	177633638630	$551 \cdot 10^{10}$	194703853226
$452 \cdot 10^{10}$	160848126971	$502 \cdot 10^{10}$	177975599097	$552 \cdot 10^{10}$	195044708509
$453 \cdot 10^{10}$	161191287194	$503 \cdot 10^{10}$	178317539923	$553 \cdot 10^{10}$	195385535193
$454 \cdot 10^{10}$	161534433032	$504 \cdot 10^{10}$	178659435867	$554 \cdot 10^{10}$	195726349627
$455 \cdot 10^{10}$	161877539236	$505 \cdot 10^{10}$	179001333344	$555 \cdot 10^{10}$	196067119388
$456 \cdot 10^{10}$	162220614553	$506 \cdot 10^{10}$	179343203262	$556 \cdot 10^{10}$	196407879245
$457 \cdot 10^{10}$	162563679498	$507 \cdot 10^{10}$	179685031468	$557 \cdot 10^{10}$	196748619154
$458 \cdot 10^{10}$	162906709169	$508 \cdot 10^{10}$	180026857317	$558 \cdot 10^{10}$	197089352128
$459 \cdot 10^{10}$	163249714268	$509 \cdot 10^{10}$	180368657311	$559 \cdot 10^{10}$	197430063663
$460 \cdot 10^{10}$	163592692077	$510 \cdot 10^{10}$	180710431905	$560 \cdot 10^{10}$	197770743547
$461 \cdot 10^{10}$	163935659989	$511 \cdot 10^{10}$	181052184728	$561 \cdot 10^{10}$	198111396558
$462 \cdot 10^{10}$	164278591350	$512 \cdot 10^{10}$	181393911476	$562 \cdot 10^{10}$	198452033395
$463 \cdot 10^{10}$	164621495465	$513 \cdot 10^{10}$	181735607644	$563 \cdot 10^{10}$	198792655849
$464 \cdot 10^{10}$	164964367966	$514 \cdot 10^{10}$	182077287959	$564 \cdot 10^{10}$	199133259931
$465 \cdot 10^{10}$	165307235443	$515 \cdot 10^{10}$	182418944564	$565 \cdot 10^{10}$	199473835744
$466 \cdot 10^{10}$	165650073995	$516 \cdot 10^{10}$	182760584708	$566 \cdot 10^{10}$	199814403456
$467 \cdot 10^{10}$	165992879523	$517 \cdot 10^{10}$	183102194747	$567 \cdot 10^{10}$	200154937236
$468 \cdot 10^{10}$	166335657132	$518 \cdot 10^{10}$	183443790483	$568 \cdot 10^{10}$	200495463127
$469 \cdot 10^{10}$	166678417618	$519 \cdot 10^{10}$	183785354647	$569 \cdot 10^{10}$	200835953437
$470 \cdot 10^{10}$	167021151370	$520 \cdot 10^{10}$	184126901158	$570 \cdot 10^{10}$	201176428961
$471 \cdot 10^{10}$	167363859472	$521 \cdot 10^{10}$	184468422118	$571 \cdot 10^{10}$	201516895747
$472 \cdot 10^{10}$	167706532636	$522 \cdot 10^{10}$	184809924015	$572 \cdot 10^{10}$	201857332816
$473 \cdot 10^{10}$	168049188517	$523 \cdot 10^{10}$	185151394459	$573 \cdot 10^{10}$	202197727802
$474 \cdot 10^{10}$	168391818231	$524 \cdot 10^{10}$	185492850429	$574 \cdot 10^{10}$	202538129085
$475 \cdot 10^{10}$	168734419733	$525 \cdot 10^{10}$	185834298906	$575 \cdot 10^{10}$	202878502933
$476 \cdot 10^{10}$	169077005780	$526 \cdot 10^{10}$	186175703033	$576 \cdot 10^{10}$	203218864752
$477 \cdot 10^{10}$	169419569068	$527 \cdot 10^{10}$	186517093016	$577 \cdot 10^{10}$	203559202990
$478 \cdot 10^{10}$	169762074315	$528 \cdot 10^{10}$	186858447989	$578 \cdot 10^{10}$	203899512777
$479 \cdot 10^{10}$	170104583073	$529 \cdot 10^{10}$	187199795745	$579 \cdot 10^{10}$	204239790060
$480 \cdot 10^{10}$	170447066694	$530 \cdot 10^{10}$	187541119457	$580 \cdot 10^{10}$	204580066580
$481 \cdot 10^{10}$	170789519178	$531 \cdot 10^{10}$	187882419840	$581 \cdot 10^{10}$	204920325312
$482 \cdot 10^{10}$	171131957185	$532 \cdot 10^{10}$	188223697005	$582 \cdot 10^{10}$	205260553100
$483 \cdot 10^{10}$	171474363203	$533 \cdot 10^{10}$	188564956085	$583 \cdot 10^{10}$	205600765730
$484 \cdot 10^{10}$	171816735058	$534 \cdot 10^{10}$	188906185764	$584 \cdot 10^{10}$	205940972436
$485 \cdot 10^{10}$	172159088446	$535 \cdot 10^{10}$	189247395968	$585 \cdot 10^{10}$	206281145677
$486 \cdot 10^{10}$	172501445056	$536 \cdot 10^{10}$	189588585443	$586 \cdot 10^{10}$	206621306540
$487 \cdot 10^{10}$	172843771881	$537 \cdot 10^{10}$	189929746258	$587 \cdot 10^{10}$	206961441960
$488 \cdot 10^{10}$	173186058474	$538 \cdot 10^{10}$	190270901650	$588 \cdot 10^{10}$	207301556829
$489 \cdot 10^{10}$	173528342165	$539 \cdot 10^{10}$	190612027654	$589 \cdot 10^{10}$	207641658170
$490 \cdot 10^{10}$	173870573446	$540 \cdot 10^{10}$	190953126302	$590 \cdot 10^{10}$	207981734215
$491 \cdot 10^{10}$	174212790220	$541 \cdot 10^{10}$	191294195107	$591 \cdot 10^{10}$	208321803297
$492 \cdot 10^{10}$	174554985222	$542 \cdot 10^{10}$	191635251660	$592 \cdot 10^{10}$	208661831879
$493 \cdot 10^{10}$	174897138199	$543 \cdot 10^{10}$	191976288777	$593 \cdot 10^{10}$	209001848271
$494 \cdot 10^{10}$	175239277386	$544 \cdot 10^{10}$	192317308442	$594 \cdot 10^{10}$	209341847526
$495 \cdot 10^{10}$	175581396501	$545 \cdot 10^{10}$	192658296815	$595 \cdot 10^{10}$	209681825130
$496 \cdot 10^{10}$	175923496965	$546 \cdot 10^{10}$	192999284353	$596 \cdot 10^{10}$	210021778129
$497 \cdot 10^{10}$	176265576237	$547 \cdot 10^{10}$	193340233672	$597 \cdot 10^{10}$	210361720982
$498 \cdot 10^{10}$	176607624598	$548 \cdot 10^{10}$	193681164450	$598 \cdot 10^{10}$	210701644389
$499 \cdot 10^{10}$	176949646367	$549 \cdot 10^{10}$	194022090027	$599 \cdot 10^{10}$	211041544021
$500 \cdot 10^{10}$	177291661649	$550 \cdot 10^{10}$	194362990637	$600 \cdot 10^{10}$	211381427039

n	$\pi(n)$	n	$\pi(n)$	n	$\pi(n)$
$601 \cdot 10^{10}$	211721277792	$651 \cdot 10^{10}$	228690644761	$701 \cdot 10^{10}$	245615785695
$602 \cdot 10^{10}$	212061122049	$652 \cdot 10^{10}$	229029572798	$702 \cdot 10^{10}$	245953864099
$603 \cdot 10^{10}$	212400947302	$653 \cdot 10^{10}$	229368476110	$703 \cdot 10^{10}$	246291925737
$604 \cdot 10^{10}$	212740739636	$654 \cdot 10^{10}$	229707370713	$704 \cdot 10^{10}$	246629968224
$605 \cdot 10^{10}$	213080533394	$655 \cdot 10^{10}$	230046244845	$705 \cdot 10^{10}$	246967999994
$606 \cdot 10^{10}$	213420303290	$656 \cdot 10^{10}$	230385078543	$706 \cdot 10^{10}$	247306015214
$607 \cdot 10^{10}$	213760058268	$657 \cdot 10^{10}$	230723919431	$707 \cdot 10^{10}$	247644018065
$608 \cdot 10^{10}$	214099778107	$658 \cdot 10^{10}$	231062739969	$708 \cdot 10^{10}$	247981996214
$609 \cdot 10^{10}$	214439487127	$659 \cdot 10^{10}$	231401539813	$709 \cdot 10^{10}$	248319951790
$610 \cdot 10^{10}$	214779167476	$660 \cdot 10^{10}$	231740339225	$710 \cdot 10^{10}$	248657916582
$611 \cdot 10^{10}$	215118832638	$661 \cdot 10^{10}$	232079110434	$711 \cdot 10^{10}$	248995838874
$612 \cdot 10^{10}$	215458483619	$662 \cdot 10^{10}$	232417853447	$712 \cdot 10^{10}$	249333758615
$613 \cdot 10^{10}$	215798125876	$663 \cdot 10^{10}$	232756574343	$713 \cdot 10^{10}$	249671653797
$614 \cdot 10^{10}$	216137751967	$664 \cdot 10^{10}$	233095290210	$714 \cdot 10^{10}$	250009547218
$615 \cdot 10^{10}$	216477357168	$665 \cdot 10^{10}$	233434002892	$715 \cdot 10^{10}$	250347417662
$616 \cdot 10^{10}$	216816941079	$666 \cdot 10^{10}$	233772688099	$716 \cdot 10^{10}$	250685276164
$617 \cdot 10^{10}$	217156495522	$667 \cdot 10^{10}$	234111336983	$717 \cdot 10^{10}$	251023109605
$618 \cdot 10^{10}$	217496032680	$668 \cdot 10^{10}$	234449975975	$718 \cdot 10^{10}$	251360926955
$619 \cdot 10^{10}$	217835547541	$669 \cdot 10^{10}$	234788607918	$719 \cdot 10^{10}$	251698723738
$620 \cdot 10^{10}$	218175058254	$670 \cdot 10^{10}$	235127223891	$720 \cdot 10^{10}$	252036515078
$621 \cdot 10^{10}$	218514552607	$671 \cdot 10^{10}$	235465829258	$721 \cdot 10^{10}$	252374287902
$622 \cdot 10^{10}$	218854013324	$672 \cdot 10^{10}$	235804399940	$722 \cdot 10^{10}$	252712045288
$623 \cdot 10^{10}$	219193444286	$673 \cdot 10^{10}$	236142954295	$723 \cdot 10^{10}$	253049788865
$624 \cdot 10^{10}$	219532882013	$674 \cdot 10^{10}$	236481500584	$724 \cdot 10^{10}$	253387509058
$625 \cdot 10^{10}$	219872297126	$675 \cdot 10^{10}$	236820018566	$725 \cdot 10^{10}$	253725222377
$626 \cdot 10^{10}$	220211701861	$676 \cdot 10^{10}$	237158529375	$726 \cdot 10^{10}$	254062910228
$627 \cdot 10^{10}$	220551076645	$677 \cdot 10^{10}$	237497029302	$727 \cdot 10^{10}$	254400585117
$628 \cdot 10^{10}$	220890419882	$678 \cdot 10^{10}$	237835503322	$728 \cdot 10^{10}$	254738235149
$629 \cdot 10^{10}$	221229761487	$679 \cdot 10^{10}$	238173962053	$729 \cdot 10^{10}$	255075883076
$630 \cdot 10^{10}$	221569102517	$680 \cdot 10^{10}$	238512407609	$730 \cdot 10^{10}$	255413505374
$631 \cdot 10^{10}$	221908403702	$681 \cdot 10^{10}$	238850814769	$731 \cdot 10^{10}$	255751128944
$632 \cdot 10^{10}$	222247686683	$682 \cdot 10^{10}$	239189228128	$732 \cdot 10^{10}$	256088720253
$633 \cdot 10^{10}$	222586940839	$683 \cdot 10^{10}$	239527608572	$733 \cdot 10^{10}$	256426299892
$634 \cdot 10^{10}$	222926180857	$684 \cdot 10^{10}$	239865980552	$734 \cdot 10^{10}$	256763857792
$635 \cdot 10^{10}$	223265403802	$685 \cdot 10^{10}$	240204337210	$735 \cdot 10^{10}$	257101409340
$636 \cdot 10^{10}$	223604605929	$686 \cdot 10^{10}$	240542681868	$736 \cdot 10^{10}$	257438941091
$637 \cdot 10^{10}$	223943785648	$687 \cdot 10^{10}$	240881019399	$737 \cdot 10^{10}$	257776449044
$638 \cdot 10^{10}$	224282974552	$688 \cdot 10^{10}$	241219299681	$738 \cdot 10^{10}$	258113950946
$639 \cdot 10^{10}$	224622141681	$689 \cdot 10^{10}$	241557588129	$739 \cdot 10^{10}$	258451445201
$640 \cdot 10^{10}$	224961273873	$690 \cdot 10^{10}$	241895851648	$740 \cdot 10^{10}$	258788925595
$641 \cdot 10^{10}$	225300394517	$691 \cdot 10^{10}$	242234113050	$741 \cdot 10^{10}$	259126391062
$642 \cdot 10^{10}$	225639499090	$692 \cdot 10^{10}$	242572364216	$742 \cdot 10^{10}$	259463843337
$643 \cdot 10^{10}$	225978591508	$693 \cdot 10^{10}$	242910578556	$743 \cdot 10^{10}$	259801270573
$644 \cdot 10^{10}$	226317652479	$694 \cdot 10^{10}$	243248787283	$744 \cdot 10^{10}$	260138683652
$645 \cdot 10^{10}$	226656712638	$695 \cdot 10^{10}$	243586989572	$745 \cdot 10^{10}$	260476073596
$646 \cdot 10^{10}$	226995745804	$696 \cdot 10^{10}$	243925170141	$746 \cdot 10^{10}$	260813454066
$647 \cdot 10^{10}$	227334761834	$697 \cdot 10^{10}$	244263318698	$747 \cdot 10^{10}$	261150824069
$648 \cdot 10^{10}$	227673756652	$698 \cdot 10^{10}$	244601451372	$748 \cdot 10^{10}$	261488181131
$649 \cdot 10^{10}$	228012732322	$699 \cdot 10^{10}$	244939572922	$749 \cdot 10^{10}$	261825521686
$650 \cdot 10^{10}$	228351698532	$700 \cdot 10^{10}$	245277688804	$750 \cdot 10^{10}$	262162841215

n	$\pi(n)$	n	$\pi(n)$	n	$\pi(n)$
$751 \cdot 10^{10}$	262500149098	$801 \cdot 10^{10}$	279346644627	$851 \cdot 10^{10}$	296157693471
$752 \cdot 10^{10}$	262837444956	$802 \cdot 10^{10}$	279683207798	$852 \cdot 10^{10}$	296493576042
$753 \cdot 10^{10}$	263174726729	$803 \cdot 10^{10}$	280019757316	$853 \cdot 10^{10}$	296829440141
$754 \cdot 10^{10}$	263511976812	$804 \cdot 10^{10}$	280356282555	$854 \cdot 10^{10}$	297165287239
$755 \cdot 10^{10}$	263849227516	$805 \cdot 10^{10}$	280692809184	$855 \cdot 10^{10}$	297501135586
$756 \cdot 10^{10}$	264186472179	$806 \cdot 10^{10}$	281029308161	$856 \cdot 10^{10}$	297836956126
$757 \cdot 10^{10}$	264523687090	$807 \cdot 10^{10}$	281365790821	$857 \cdot 10^{10}$	298172766089
$758 \cdot 10^{10}$	264860883089	$808 \cdot 10^{10}$	281702266186	$858 \cdot 10^{10}$	298508555121
$759 \cdot 10^{10}$	265198066615	$809 \cdot 10^{10}$	282038732056	$859 \cdot 10^{10}$	298844339427
$760 \cdot 10^{10}$	265535241952	$810 \cdot 10^{10}$	282375171661	$860 \cdot 10^{10}$	299180106058
$761 \cdot 10^{10}$	265872387443	$811 \cdot 10^{10}$	282711609598	$861 \cdot 10^{10}$	299515865435
$762 \cdot 10^{10}$	266209528884	$812 \cdot 10^{10}$	283048037431	$862 \cdot 10^{10}$	299851618704
$763 \cdot 10^{10}$	266546651105	$813 \cdot 10^{10}$	283384437149	$863 \cdot 10^{10}$	300187341487
$764 \cdot 10^{10}$	266883766717	$814 \cdot 10^{10}$	283720835629	$864 \cdot 10^{10}$	300523068997
$765 \cdot 10^{10}$	267220850334	$815 \cdot 10^{10}$	284057215362	$865 \cdot 10^{10}$	300858757405
$766 \cdot 10^{10}$	267557944858	$816 \cdot 10^{10}$	284393576877	$866 \cdot 10^{10}$	301194460343
$767 \cdot 10^{10}$	267895012252	$817 \cdot 10^{10}$	284729924748	$867 \cdot 10^{10}$	301530144529
$768 \cdot 10^{10}$	268232071500	$818 \cdot 10^{10}$	285066247539	$868 \cdot 10^{10}$	301865820404
$769 \cdot 10^{10}$	268569118961	$819 \cdot 10^{10}$	285402566544	$869 \cdot 10^{10}$	302201471940
$770 \cdot 10^{10}$	268906152117	$820 \cdot 10^{10}$	285738881494	$870 \cdot 10^{10}$	302537108851
$771 \cdot 10^{10}$	269243161373	$821 \cdot 10^{10}$	286075172221	$871 \cdot 10^{10}$	302872738734
$772 \cdot 10^{10}$	269580142667	$822 \cdot 10^{10}$	286411447923	$872 \cdot 10^{10}$	303208347931
$773 \cdot 10^{10}$	269917120224	$823 \cdot 10^{10}$	286747706445	$873 \cdot 10^{10}$	303543949197
$774 \cdot 10^{10}$	270254093571	$824 \cdot 10^{10}$	287083967008	$874 \cdot 10^{10}$	303879546236
$775 \cdot 10^{10}$	270591037559	$825 \cdot 10^{10}$	287420205390	$875 \cdot 10^{10}$	304215123662
$776 \cdot 10^{10}$	270927970769	$826 \cdot 10^{10}$	287756433268	$876 \cdot 10^{10}$	304550681623
$777 \cdot 10^{10}$	271264889348	$827 \cdot 10^{10}$	288092636142	$877 \cdot 10^{10}$	304886221861
$778 \cdot 10^{10}$	271601799369	$828 \cdot 10^{10}$	288428831589	$878 \cdot 10^{10}$	305221750689
$779 \cdot 10^{10}$	271938681593	$829 \cdot 10^{10}$	288765025983	$879 \cdot 10^{10}$	305557291866
$780 \cdot 10^{10}$	272275559525	$830 \cdot 10^{10}$	289101209285	$880 \cdot 10^{10}$	305892791866
$781 \cdot 10^{10}$	272612434488	$831 \cdot 10^{10}$	289437370676	$881 \cdot 10^{10}$	306228288735
$782 \cdot 10^{10}$	272949280138	$832 \cdot 10^{10}$	289773517505	$882 \cdot 10^{10}$	306563782896
$783 \cdot 10^{10}$	273286116126	$833 \cdot 10^{10}$	290109645702	$883 \cdot 10^{10}$	306899250550
$784 \cdot 10^{10}$	273622929571	$834 \cdot 10^{10}$	290445761957	$884 \cdot 10^{10}$	307234713722
$785 \cdot 10^{10}$	273959728234	$835 \cdot 10^{10}$	290781864888	$885 \cdot 10^{10}$	307570157417
$786 \cdot 10^{10}$	274296529203	$836 \cdot 10^{10}$	291117947137	$886 \cdot 10^{10}$	307905604753
$787 \cdot 10^{10}$	274633306068	$837 \cdot 10^{10}$	291454028751	$887 \cdot 10^{10}$	308241028083
$788 \cdot 10^{10}$	274970064256	$838 \cdot 10^{10}$	291790095436	$888 \cdot 10^{10}$	308576434573
$789 \cdot 10^{10}$	275306809536	$839 \cdot 10^{10}$	292126147116	$889 \cdot 10^{10}$	308911839233
$790 \cdot 10^{10}$	275643544693	$840 \cdot 10^{10}$	292462199848	$890 \cdot 10^{10}$	309247216816
$791 \cdot 10^{10}$	275980253044	$841 \cdot 10^{10}$	292798212679	$891 \cdot 10^{10}$	309582581121
$792 \cdot 10^{10}$	276316969024	$842 \cdot 10^{10}$	293134213734	$892 \cdot 10^{10}$	309917947401
$793 \cdot 10^{10}$	276653648332	$843 \cdot 10^{10}$	293470219321	$893 \cdot 10^{10}$	310253288649
$794 \cdot 10^{10}$	276990320475	$844 \cdot 10^{10}$	293806192711	$894 \cdot 10^{10}$	310588621114
$795 \cdot 10^{10}$	277326971562	$845 \cdot 10^{10}$	294142152978	$895 \cdot 10^{10}$	310923954796
$796 \cdot 10^{10}$	277663616294	$846 \cdot 10^{10}$	294478108014	$896 \cdot 10^{10}$	311259263348
$797 \cdot 10^{10}$	278000257081	$847 \cdot 10^{10}$	294814053640	$897 \cdot 10^{10}$	311594551034
$798 \cdot 10^{10}$	278336884667	$848 \cdot 10^{10}$	295149989855	$898 \cdot 10^{10}$	311929831566
$799 \cdot 10^{10}$	278673481953	$849 \cdot 10^{10}$	295485900910	$899 \cdot 10^{10}$	312265103220
$800 \cdot 10^{10}$	279010070811	$850 \cdot 10^{10}$	295821814466	$900 \cdot 10^{10}$	312600354108

n	$\pi(n)$	n	$\pi(n)$	n	$\pi(n)$
$901 \cdot 10^{10}$	312935616309	$935 \cdot 10^{10}$	324326675369	$969 \cdot 10^{10}$	335703843585
$902 \cdot 10^{10}$	313270841946	$936 \cdot 10^{10}$	324661512052	$970 \cdot 10^{10}$	336038264769
$903 \cdot 10^{10}$	313606067157	$937 \cdot 10^{10}$	324996318173	$971 \cdot 10^{10}$	336372648585
$904 \cdot 10^{10}$	313941266729	$938 \cdot 10^{10}$	325331105431	$972 \cdot 10^{10}$	336707058596
$905 \cdot 10^{10}$	314276474927	$939 \cdot 10^{10}$	325665885320	$973 \cdot 10^{10}$	337041458081
$906 \cdot 10^{10}$	314611668924	$940 \cdot 10^{10}$	326000647688	$974 \cdot 10^{10}$	337375826503
$907 \cdot 10^{10}$	314946832423	$941 \cdot 10^{10}$	326335420356	$975 \cdot 10^{10}$	337710190696
$908 \cdot 10^{10}$	315281977559	$942 \cdot 10^{10}$	326670165889	$976 \cdot 10^{10}$	338044529198
$909 \cdot 10^{10}$	315617130081	$943 \cdot 10^{10}$	327004910769	$977 \cdot 10^{10}$	338378859442
$910 \cdot 10^{10}$	315952260435	$944 \cdot 10^{10}$	327339631976	$978 \cdot 10^{10}$	338713178503
$911 \cdot 10^{10}$	316287384334	$945 \cdot 10^{10}$	327674339039	$979 \cdot 10^{10}$	339047492382
$912 \cdot 10^{10}$	316622483923	$946 \cdot 10^{10}$	328009028448	$980 \cdot 10^{10}$	339381817251
$913 \cdot 10^{10}$	316957582204	$947 \cdot 10^{10}$	328343721109	$981 \cdot 10^{10}$	339716100374
$914 \cdot 10^{10}$	317292680200	$948 \cdot 10^{10}$	328678391458	$982 \cdot 10^{10}$	340050387015
$915 \cdot 10^{10}$	317627755961	$949 \cdot 10^{10}$	329013051908	$983 \cdot 10^{10}$	340384657291
$916 \cdot 10^{10}$	317962815473	$950 \cdot 10^{10}$	329347709593	$984 \cdot 10^{10}$	340718923086
$917 \cdot 10^{10}$	318297874246	$951 \cdot 10^{10}$	329682336857	$985 \cdot 10^{10}$	341053167378
$918 \cdot 10^{10}$	318632914858	$952 \cdot 10^{10}$	330016965946	$986 \cdot 10^{10}$	341387412852
$919 \cdot 10^{10}$	318967942278	$953 \cdot 10^{10}$	330351582143	$987 \cdot 10^{10}$	341721638611
$920 \cdot 10^{10}$	319302948414	$954 \cdot 10^{10}$	330686182758	$988 \cdot 10^{10}$	342055848537
$921 \cdot 10^{10}$	319637958244	$955 \cdot 10^{10}$	331020773302	$989 \cdot 10^{10}$	342390044826
$922 \cdot 10^{10}$	319972940843	$956 \cdot 10^{10}$	331355358569	$990 \cdot 10^{10}$	342724224549
$923 \cdot 10^{10}$	320307935236	$957 \cdot 10^{10}$	331689931852	$991 \cdot 10^{10}$	343058400378
$924 \cdot 10^{10}$	320642901872	$958 \cdot 10^{10}$	332024496516	$992 \cdot 10^{10}$	343392568900
$925 \cdot 10^{10}$	320977854538	$959 \cdot 10^{10}$	332359037746	$993 \cdot 10^{10}$	343726740721
$926 \cdot 10^{10}$	321312783265	$960 \cdot 10^{10}$	332693561966	$994 \cdot 10^{10}$	344060885763
$927 \cdot 10^{10}$	321647716577	$961 \cdot 10^{10}$	333028092304	$995 \cdot 10^{10}$	344395027557
$928 \cdot 10^{10}$	321982619150	$962 \cdot 10^{10}$	333362587598	$996 \cdot 10^{10}$	344729151839
$929 \cdot 10^{10}$	322317523994	$963 \cdot 10^{10}$	333697093196	$997 \cdot 10^{10}$	345063273171
$930 \cdot 10^{10}$	322652410741	$964 \cdot 10^{10}$	334031591394	$998 \cdot 10^{10}$	345397374635
$931 \cdot 10^{10}$	322987287822	$965 \cdot 10^{10}$	334366057075	$999 \cdot 10^{10}$	345731460823
$932 \cdot 10^{10}$	323322150132	$966 \cdot 10^{10}$	334700516130	$1000 \cdot 10^{10}$	346065536839
$933 \cdot 10^{10}$	323657005401	$967 \cdot 10^{10}$	335034974233		
$934 \cdot 10^{10}$	323991836237	$968 \cdot 10^{10}$	335369421494		

8 最後に

ここまで，いくつかのエラトステネス流の篩（表から素数倍の数を除いていく）を使って素数表を作る説明をしてきました。

素数表作成の動機は，古くは素因数分解の補助としてのものでした。素因数分解は，数を調べるうえで基本的なものですが，比較的大きな数を手計算で素因数分解するのは中々難しいものです。例えば，1234567 を手計算で素因数分解するには，少し根気が必要です。このような計算を補助するものとして，まず，最小素因数表が作られました。最小素因数表は，各数の最小素因数を表にしたもので，これがあれば，完全な素因数分解は簡単に可能です。ただ，最小素因数表は，各々の数について表にする必要がありますから，大きな数迄の表を作ると，それだけ大きな表が必要です。そしてそれより小さい表で，計算の補助となるものが素数表でした。

例えば，上の例で 1234567 を素因数分解するとします。1234567 以上の最小素因数表があれば，その表から最小素因数を求めて，それで割り，更にその割った結果の最小素因数を表から調べることで，1234567 の素因数分解が完成します。

最小素因数表が無くて，素数表がある場合は，少し手間ですが，素数表にある小さい素数から順次割っていきます。今の場合，$2, 3, 5$ で割り切れないことはすぐに分かりますから，実際には，$7, 11, 13, 17, 19, \ldots$ と順次割って行きます。すると，7 から数えて，28 回目で 127 で割り切れることが分かります。商は 9721 ですが，$9271 < 100^2$ ですから，9271 も素数であることが分かり，1234567 の素因数分解 $= 127 \cdot 9271$ が完成します。

素数表が無い場合だと，7 以後の奇数で割って行くことになり，61 回の割り算で 127 を見つけることができます。

一般に，素数は大きくなるにつれてまばらになりますから，素数表を利用すれば，素数表が無い場合に比べて，素因数分解はより効率的になります。

このように，元々は素数表は素因数分解の補助道具という実用的な目的で作成されました。しかし，現在では，コンピュータを使えば Long の範囲での素因数分解は，素数表が無くても比較的簡単に可能です[*99][*100]。ですから，現在では素数表の，手計算での素因数分解の補助としての役割は終わったと言えます。

しかし，それでも素数表の重要性は変わりません。素数表は素数研究の原点のデータベースとも言えるからです。多くの予想は，素数表を基になされ，検証され，証明への動機づけを与えました。素数表があれば，それを単に眺めるだけでも，素数についての多くの性質や情報を与えてくれます。そして大きな素数表であればあるほど，多くの情報を与えてくれます。ですから，より大きな素数表を求める挑戦は自然なものと言えます。

しかし，一方，大きな素数表は余りにも大きいな保存スペースを必要とするので，ファイルとして保存して利用するには適していません。しかし，素数表から得たい情報を得るために，素数表全体が一度に保存してある必要は，必ずしもありません。表１，２を作成したときのように，区分法を用いれば，部分的に素数表を作成し，それらの計算結果をまとめることで，求める情報の殆どを得ることができるでしょう。大きな素数表を効率的に短時間で作るということの効用は素数表自体を求めるということではなく，むしらそれを用いて種々の素数の性質データを収集することと言えるかもしれません。

ここで説明したものは，単純なものですが，素数の性質に興味ある人々が簡単に比較的大きな素数表を扱えるということに意味があると思っています。計算機の高速化と，もう少し使用メモリーが増えれば，10^{14}，あるいはさらに進んで，10^{15}，10^{16} 以下の素数表の作成はここで説明した技法でも可能でしょう。

[*99] tbasic では，最小素因数を返す関数 PFactor もあります。

[*100] コンピュータでの素因数分解は，実際には素数表を使って，上の計算と同様な処理を高速に行うことが多いようです。