PRIMTAL

ET GIER DEMONSTRATIONSPROGRAM

Jørgen Kjær

Haldor Topsøe, Vedbæk, Danmark

1967

Copy No.

Haldor Topsøe, Chemical Engineers

Vedbæk, Denmark

FORORD

I fortsættelse af den tidligere udsendte lille beskrivelse af tændstikspillet NIM følger hermed et andet GIER demonstrationsprogram, som beregner primtal og primfaktorer.

Det væsentligste formaal med et program til beregning af primtal er at demonstrere maskinens store regnehastighed, idet primtallene beregnes ved simpel division af de foreliggende tal med et stort antal divisorer. Programmet er som helhed skrevet i ALGOL, men den centrale del, der udfører de mange divisioner, er skrevet i GIER maskinkode. Medens NIM-programmet var skrevet i GIER ALGOL III, er nærværende primtalsprogram skrevet i GIER ALGOL 4, som direkte tillader en blanding af ALGOL og maskinkode. Jeg tror, at der for den yngre generation af programmører, som ikke har lært maskinkode før de lærte ALGOL, er et behov for en pædagogisk assistance i tilegnelsen af denne programmeringsform, der er meget væsentlig for fremstilling af effektive programmer, især til egentlige databehandlingsopgaver. Som følge heraf har jeg givet en meget udførlig forklaring paa ordrerne i de 14 celler, der indeholder maskinkoden.

Da de forskellige administrationsprocedurer til talindlæsning, besvarelse med ja eller nej, o.s.v., har været udførligt omtalt i beskrivelsen af NIM-spillet, er de kun nævnt ganske kort i nærværende beskrivelse. Men da der kan være mindre forskelle betinget af overgangen fra GIER ALGOL III til GIER ALGOL 4, er proceduredeklarationerne medtaget fuldt ud her.

Beskrivelsen vil senere udkomme paa engelsk.

Virum, Marts 1967

Jørgen Kjær.

INDHOLDSFORTEGNELSE

1. INDLEDNING

Beregning af primtal har altid været et yndet emne indenfor den matematiske underholdningsbranche, af den simple grund, at der ikke findes nogen matematisk metode til paa simpel vis at komme fra eet primtal til det næste i rækken. For ganske specielle tal har man særlige testprocedurer, som kan afgøre om tallet er et primtal eller ej, uden at det er nødvendigt at foretage et stort antal regneoperationer, men for det almindelige tilfælde er der ingen genvej, man maa foretage en masse divisioner for at se, om tallet er et primtal.

Da divisionshastigheden for en elektronregnemaskine er meget stor (ca. 250 mikrosekunder i GIER), har vi her en mulighed for at demonstrere maskinens store regnehastighed ved at beregne primtal eller primfaktorer. I det foreliggende demonstrationsprogram opgiver man en nedre og øvre grænse for de tal, som maskinen skal undersøge, og programmet vil da enten finde alle primtal i dette omraade eller opløse alle tallene i omraadet i primfaktorer.

Programmet behandler kun saadanne heltal, som kan rummes i en enkelt celle, d.v.s. den øvre grænse for tallene er 549755813887 (= $2^{39} - 1$).

I kapitel 2 beskrives den anvendte divisionsmetode, der er en mellemting mellem den helt simple division og den klassiske Eratosthenes si.

I kapitel 3 omtales forskellige administrationsprocedurer til kontrol af sideskift, til udskrivning af tekst paa de fire forskellige sprog (Dansk, Engelsk, Fransk, Tysk), som maskinen kan arbejde i, og til trykning af tal.

De egentlige primtalsprocedurer beskrives i kapitel 4. Selve programmet omtales i kapitel 5, og der gives referencer i kapitel 6. Endelig giver kapitel 7 en udskrift af ALGOL-programmet og et eksempel paa en udskrift fra en kørsel med programmet.

2. PRIMTAL OG PRIMFAKTORER

2.1. Alment.

Et primtal er et positivt, helt tal, der kun kan divideres med tallet selv og med 1. Division med alle andre positive heltal giver en rest forskellig fra nul. Tallet 1 regnes ikke for et primtal.

De første primtal er:

2, 3, 5, 7, 11, 13, 17, 19, 23, ...

Primtallenes egenskaber beskrives i de fleste lærebøger i talteori. En mere populær fremstilling af primtallene findes f. eks. i Beiler (1964). Blandt de mange mærkelige sætninger, der gælder for primtal, kan nævnes:

1. Der er uendelig mange primtal.

2. Der er uendelig mange primtalspar, d.v.s. to primtal med en forskel paa 2:

..... 101, 103, ... 107, 109, ... 137, 139, ...

3. Ethvert lige heltal kan skrives som en sum af to primtal. Dette er Goldbachs teorem, som aldrig har kunnet bevises eller modbevises.

4. Hvis et positivt, helt tal ikke er et primtal, kan det skrives som et produkt af primfaktorer paa een og kun een maade, f.eks.:

$$765 = 3 \times 5 \times 17$$

Der findes ingen regel, som paa simpel vis tillader at beregne primtal nr. N, bortset fra slavemetoden med division. Men man kan vise, at den gennemsnitlige afstand mellem to paa hinanden følgende primtal i nærheden af tallet N er ca. ln(N). For N = 123456789 er ln(N) = ca. 19, saaledes at der gennemsnitligt er 19 mellem to paa hinanden følgende primtal.

2.2. Eratosthenes si.

En simpel metode til beregning af primtalstabeller indenfor et givet talomraade er angivet allerede af Eratosthenes fra den græske oldtid. Man skriver først alle tallene i omraadet op, f.eks.:

101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
111, 112, 113, 114, 115, 116, 117, 118, 119, 120,
. . .

Derefter fjerner man alle de lige tal, d.v.s. de tal som kan divideres med to. Tilbage bliver:

101	103	105	107	109
111	113	115	117	119

Derefter fjernes alle multipla af 3, altsaa hvert tredje tal. Tilbage bliver:

101	103	107	109
	113		119

Paa denne maade fortsætter vi med at fjerne multipla af 5, 7, 11, o.s.v. Tilbage efter sigteprocessen bliver primtallene i omraadet:

101	103	107	109
	113		

Heraf navnet: Eratosthenes si. Metoden kan let programmeres og har den fordel, at der slet ikke kræves division.

2.3. Nærværende metode.

Hvis man kun skal undersøge faa tal for, om de er primtal eller ej, er Eratosthenes si for langsom, idet der gaar for lang tid med at finde det første multiplum, som skal fjernes i tabellen. Dette kræver en division for hver faktor, som skal undersøges. Det er da hurtigere at bruge simpel division.

Skal vi undersøge et foreliggende tal, f.eks. 113, skulle vi strengt taget efter definitionen paa primtal prøve alle divisiorer:

2, 3, 4, 5, ... 112

altsaa ialt 111 divisorer. Men i praksis behøver vi ikke saa mange divisorer. For det første behøver vi ikke at gaa saa højt som 112. Undersøger vi nemlig i øjeblikket divisoren D, og er der ingen divisorer lavere end D, behøver vi ikke at lade D blive større, end at der gælder:

$D^2 \leq 113$.

Saasnart $D^2 > 113$, behøver vi ikke at undersøge flere divisorer. Da kvadratroden af 113 ligger mellem 10 og 11, er 10 den højeste divisor, vi skal prøve. Divisorerne er altsaa:

2, 3, 4, 5, 6, 7, 8, 9, 10,

Altsaa 9 divisorer. Men for det andet behøver vi kun at bruge primtal som divisorer, altsaa de 4 divisorer:

2, 3, 5, 7

Nu vil det i praksis være besværligt at teste hver eneste divisor for at se, om det er et primtal, saa vi kan vælge et kompromis ved at fjerne de fleste ikke-primtalsdivisorer ved hjælp af Eratosthenes si. Fjernelse af 2-multipla giver følgende divisorer:

2, 3, 5, 7, 9

Fjerner vi yderligere 3-multipla, er der kun primtalsdivisorer tilbage:

2, 3, 5, 7

I den anvendte divisionsmetode i maskinkode fjernes alle multipla af 2, 3 og 5 fra divisorrækken. Proceduren beskrives nøjere i afsnit 4.1.

3. ADMINISTRATIONSPROCEDURER

De fleste af de nedennævnte procedurer er forklaret i NIM-beskrivelsen, se Kjær (1966).

3.1. Sideskift.

Udskrift paa papirbaner med 70 linier pr. side varetages af de to procedurer, NEW PAGE og LINE. Desuden bruges den globale variable: integer linerest, der angiver, hvormange frie linier, der er tilbage paa siden.

3.1.1. Proceduren NEW PAGE. Deklarationen herfor er:

```
procedure NEW PAGE;
begin
   for linerest := linerest - 1 while linerest ≥ 0, 69 do writecr;
   writechar(72)
end NEW PAGE;
```

Proceduren er ændret fra GIER ALGOL III, idet den egentlige ordre for ny linie:

```
writecr;
```

skrives paa samme maade i GIER ALGOL 4. De rigtige ydre enheder skal være valgt i forvejen ved en select-ordre:

```
select(17);
```

der staar i begyndelsen af programmet (se kapitel 5) og som vælger input og output fra skrivemaskinen.

I NEW PAGE har vi tilføjet ordren:

```
writechar(72);
```

efter sideskiftet, som giver ny side paa linieskriveren, hvis denne skulle være tilkoblet via krydsfeltet.

3.1.2. Proceduren LINE. Deklarationen er:

```
procedure LINE;
if linerest < 8 then NEWPAGE
else
begin
   linerest := linerest -1;
   writecr
end LINE;
```

Den er ændret fra GIER ALGOL III.

3.2. Spørgsmaal.

Til udveksling af oplysninger med maskinen bruges de fire procedurer: WRITE TEXT, SELECT LANGUAGE, ASK NUMBER og QUESTION. Desuden bruges den globale variable:

```
integer lang;
```

De fire værdier (1, 2, 3, og 4), som lang skal kunne antage, skal svare til de fire sprog (Dansk, Engelsk, Fransk og Tysk), som vi ønsker at programmet skal kunne arbejde i.

3.2.1. Proceduren WRITE TEXT. Deklarationen herfor er:

```
procedure WRITE TEXT(dan, eng, fr, ger);
string dan, eng, fr, ger;
writetext(case lang of(dan, eng, fr, ger));
```

I GIER ALGOL 4 har vi sætningen:

```
writetext(case lang of (dan, eng, fr, ger));
```

som er meget kortere end den tilsvarende formulering i GIER ALGOL III:

```
writetext(if lang = 1 then dan
     else(if lang = 2 then eng
     else(if lang = 3 then fr else ger);
```

Hvis lang er lig 1, bruges den første formelle parameter, dan, som er den danske tekststreng, som programmet skal trykke. Er lang = 2 faas den engelske tekststreng, o.s.v.

Bemærk, at ALGOL III sætningen ikke er tilladt i GIER ALGOL 4.

3.2.2. Proceduren SELECT LANGUAGE. Deklarationen er:

```
procedure SELECT LANGUAGE;
begin
   LINE;
   writetext(
   {<Select language: d: danish, e: english, f: french, g: german.: });
   lang := lyn - 51;
   if lang < 1 then lang := 1;
   if lang > 4 then lang := 4;
   LINE;
   WRITE TEXT(
   {<Dansk},
   {<English},
   {<Francais},
   {<Deutsch});
   LINE
end SELECT LANGUAGE;
```

Proceduren er ændret fra udgaven i GIER ALGOL III, bortset fra sætningen:

```
lang := typechar - 51;
```

der nu skrives som:

```
lang := lyn - 51;
```

idet typechar ikke længere findes i ALGOL 4.

3.2.3. Proceduren ASK NUMBER. Deklarationen er:

```
integer procedure ASK NUMBER(dan, eng, fr, ger);
string dan, eng, fr, ger;
begin
   LINE;
   WRITE TEXT(dan, eng, fr, ger);
   writetext(‹: ›);
   ASK NUMBER := read integer
end ASK NUMBER;
```

Der er indført to ændringer fra versionen i GIER ALGOL III. For det første er proceduren gjort til en integer procedure istedet for en real procedure, fordi vi kun skal bruge den til indlæsning af heltal. For det andet er den tidligere sætning:

```
ASK NUMBER := typein;
```

erstattet med:

```
ASK NUMBER := read integer;
```

Den specielle skrivemaskininputprocedure, typein, findes ikke længere i GIER ALGOL 4, hvor input- og outputmedium vælges med select-proceduren. Den nye procedure, read integer, læser heltal fra det valgte medium.

3.2.4. Proceduren QUESTION. Deklarationen herfor er:

```
boolean procedure QUESTION(dan, eng, fr, ger);
string dan, eng, fr, ger;
begin
   integer symb;
AGAIN:LINE;
   WRITE TEXT(dan, eng, fr, ger);
   writetext(‹: ›);
   symb := lyn;
   if symb = 37 then
   WRITE TEXT(‹ej›, ‹o›, ‹on›, ‹ein›)
   else
   if symb = 24 ∨ symb = 33 ∨ symb = 38 then
```

```
begin
   lang := if symb = 24 then 2
   else if symb = 38 then 3
   else if lang ≤ 2 then 1 else 4;
   WRITE TEXT(‹<a›, ‹<es›, ‹<ui›, ‹<a›)
end if yes
else
begin
   LINE;
   WRITE TEXT(
   ‹<Det forstaar jeg ikke›,
   ‹<I do not understand this›,
   ‹<Je ne comprends pas cela›,
   ‹<Das verstehe ich nicht›);
   go to AGAIN
end if nonsense;
QUESTION := symb ≠ 37
end QUESTION;
```

Ogsaa her er den eneste ændring, at proceduren typechar er erstattet af proceduren lyn.

3.3. Talttrykning.

I langt de fleste tilfælde vil man være fuldt ud tilfreds med at trykke resultater fra en beregning med de eksisterende standardprocedurer hertil. I GIER ALGOL 4 kan et saadant procedurekald se saaledes ud:

```
write(‹-dddddd›, N);
```

Her trykkes tallet N med 6 heltalscifre og fortegn.

I primtalprogrammet skal vi kunne trykke tal med op til 12 heltalscifre, og vi maa derfor bruge et layout som ‹-ddddddddddd›, hvis vi vil trykke alle tal med samme layout. Det er daarlig økonomi at bruge et saa stort layout til trykning af tal med meget faa cifre. Programmet er derfor indrettet med en special procedure, PRINT LEFT, som trykker et venstrenormaliseret tal, d.v.s. der fyldes ikke op med mellemrum før selve tallet. De trykte tal kommer til at staa saaledes:

123 124 125 126 127
1234 1235 1236 1237
12345 12346 12347

Proceduren tæller ogsaa, hvor mange cifre, der faktisk er trykt i tallet.

3.3.1. Proceduren PRINT LEFT. Deklarationen er:

```
procedure PRINT LEFT(number, places);
value number;
integer number, places;
begin
   boolean started;
   integer DIVISOR, symbol, count;
   DIVISOR := modulus;
   places := 0;
   started := false;
   for count := 1 step 1 until 12 do
   begin
      symbol := number:DIVISOR;
      number := numbermodDIVISOR;
      if symbol ≠ 0 then started := true;
      if symbol = 0 then symbol := 16;
      if started then
      begin
         writechar(symbol);
         places := places + 1
      end if started;
      DIVISOR := DIVISOR:10
   end for symbol
end PRINT LEFT;
```

De to formelle parametre er:

integer number: Det tal, som skal trykkes.
integer places: Antallet af cifre i tallet. Dette tælles og afleveres af proceduren.

De enkelte cifre beregnes og trykkes saaledes. Først sættes den lokale variable, DIVISOR, lig med en global variabel, modulus, der i dette program har værdien: 100000000000. Vi sætter parametren: places til nul og den logiske variable: started til falsk.

Lad os antage, at tallet, der skal trykkes, er: 123456789. Vi gennemløber en for-sætning, hvor den lokale variable: count tælles op fra 1 til 12.

For hvert gennemløb af for-sætningen beregnes først det ønskede ciffer ved division:

```
symbol := number:DIVISOR;
```

og number (som er kaldt ved value), sættes lig resten ved divisionen:

```
number := number mod DIVISOR;
```

Vort eksempel giver for count = 1 at symbol bliver nul, og number er uændret. Derefter følger sætningen:

```
if symbol ≠ 0 then started := true;
```

Meningen er, at lige saa snart der dukker et ciffer op, som er forskelligt fra nul, skal started sættes til sand som tegn paa at tallet er begyndt, og at vi nu maa trykke cifrene. Den næste sætning er:

```
if symbol = 0 then symbol := 16;
```

som er nødvendig, fordi et nul har karakterværdien 16. Det havde iøvrigt været lidt fiksere at skrive de to sidste sætninger som een:

```
if symbol ≠ 0 then started := true else symbol := 16;
```

Vi er nu klar til at trykke cifferet, hvis det ellers skal trykkes. Det sker med sætningen:

```
if started then
begin
   writechar(symbol);
   places := places + 1
end if started;
```

Trykningen sker med writechar (symbol) og der tælles een frem i places.

Sidste sætning i den store for-sætning er division af DIVISOR med 10. For count = 1 bliver DIVISOR altsaa 10000000000. I vort eksempel er DIVISOR > number for count = 1, 2 og 3. Først for count = 4 har vi:

number: 123456789
DIVISOR: 100000000

Divisionen giver nu:

symbol: 1
number: 23456789

og vi faar trykt 1-tallet. For count = 5 er DIVISOR igen blevet 10 gange mindre; vi faar klippet 2-tallet fra til trykning, resten bliver: 3456789, o.s.v. indtil hele tallet er trykt.

3.3.2. Proceduren PRINT PRIME. Deklarationen er:

```
procedure PRINT PRIME(p);
value p;
integer p;
begin
   prime count := prime count + 1;
   writechar(0);
   PRINT LEFT(p, places);
   place rest := place rest - places - 1;
   if place rest < 0 then
   begin
      LINE;
      place rest := 65
   end if new line
end PRINT PRIME;
```

Denne procedure bruges, naar programmet kun beregner primtal, ikke primfaktorer. Proceduren tæller een frem i en global variabel, prime count, der tæller det samlede antal primtal, som er fundet. Derefter

trykkes et mellemrum med writechar(0), og primtallet, p, trykkes med PRINT LEFT(p, places).

Nu subtraheres places + 1 fra en anden global variabel, place rest, som tæller det samlede antal anslag paa en linie. Hvis place rest bliver negativ, trykkes en ny linie med LINE, og place rest gives startværdien 65. Ved beregningens begyndelse maa place rest ogsaa sættes til 65, ligesom prime count maa være sat til nul.

4. PRIMTALSPROCEDURER

Programmet indeholder tre primtalsprocedurer: Et stykke maskinkode og de to ALGOL-procedurer NEXT PRIME og PRINT FACTORS. Begge ALGOL-procedurer kalder maskinkoden internt.

4.1. Maskinkoden.

Maskinkoden er vist paa næste side. Den dividerer et foreliggende tal, test number, med stadig større divisorer, som forklaret i afsnit 2.3. Hvis en division gaar op, hoppes ud fra proceduren, efter at en logisk variabel, divisible, er sat til sand. Hvis divisionerne ikke gaar op, fortsættes med øgningen af divisoren, indtil en vis grænse, limit, er naaet. Der hoppes da ud med divisible sat til falsk. Det foreliggende tal har da været et primtal.

```
core code code1, divisible, test number, limit, divisor;
2, 46;
2, 46;
2, 44;
2, 44;
2, 44;
grn r+28          ; 0  divisor := 0
pa  r+3 t +9      ; 1  set address
pm  a3  ,gm  r+24 ; 2  move test number
pm  a4  ,gm  r+24 ; 3  move limit
arn r+0 t +1 IPA  ; 4  fetch increment
ac  r+23,pmn r+21 ; 5  add to divisor, M := test n.
pa  r-2 t+13 LPA  ; 6  reset address
arn r+20,sr  r+21 ; 7  limit - divisor
hhn r+3      LT   ; 8  finished, not divisible
dln r+19X         ; 9  /divisor
hv  r-6      NZ   ; 10 not divisible
arn r-1 ,gr  a2   ; 11 divisible, store divisible
pm  r+16,gm  a5   ; 12 move divisor
hr  s+1           ; 13 exit
qq  2.39          ; 14 2
qq  1.39          ; 15 3
qq  2.39          ; 16 5
qq  2.39          ; 17 7
qq  4.39          ; 18 11 41  71
qq  2.39          ; 19 13 43  73
qq  4.39          ; 20 17 47  77
qq  2.39          ; 21 19 49  79
qq  4.39          ; 22 23 53  83
qq  6.39          ; 23 29 59  89
qq  2.39          ; 24 31 61  91
qq  6.39,         ; 25 37 67  97
qq                ; 26 test number
qq                ; 27 limit
qq                ; 28 divisor
e                 ;
core code code2;
2, 46;
qq  0
e;
code2 := boolean(integer code1 + integer 10 2);
```

Det egentlige indhold af cellerne i maskinkoden er noteret i de 29 celler mærket fra 0 til 28. Før disse celler staar der en slags deklaration af maskinkoden:

```
core code code1, divisible, test number, limit, divisor;
2, 46;
2, 46;
2, 44;
2, 44;
2, 44;
```

ALGOL-ordene core code betyder, at der nu kommer et stykke maskinkode. Betegnelsen core viser, at maskinkoden skal placeres fast i lageret (i stakken af det kørende ALGOL-system). Ved indhoppet til den blok, i hvilken core code er deklareret (her den yderste blok), vil det kørende system anbringe de 29 maskinkodeordrer i stakken, hvor de vil staa uændret, indtil blokken nedlægges igen. Der findes ogsaa en anden slags maskinkode i GIER ALGOL 4, som blot betegnes code. Denne anbringes ikke i stakken, men indgaar i programsegmenterne paa lige fod med de øvrige ALGOL-sætninger.

De fem navne efter ordet core code: code1, divisible, test number, limit og divisor har følgende betydning. Det er navnene paa de variable, som maskinkoden skal kunne arbejde paa. I det foreliggende tilfælde er code1 og divisible af typen boolean medens de tre sidste er af typen integer. De er alle deklareret i den yderste blok. Noget af denne information indeholdes i de fem talgrupper, som staar i de næste fem linier efter core code linien:

```
2, 46;
2, 46;
2, 44;
2, 44;
2, 44;
```

De fem talsæt svarer til de fem navne, saaledes at 2-tallet betyder en variabel i den yderste blok, og som skal adresseres absolut. Tallet 46 betyder en simpel variabel af typen boolean og 44 en simpel variabel af typen integer.

Efter de indledende deklarationslinier følger selve maskinkoden. Den er opdelt i tre dele:

Celle 0-13 indeholder de egentlige ordrer i maskinsprog. De forklares nøjere nedenfor.

Celle 14-25 indeholder de talkonstanter, der skal bruges til Eratosthenes si. Betegnelsen qq 2.39 betyder et 2-tal med enheden i bit 39 af cellen, altsaa et 2-tal af typen integer. De tolv talkonstanter er:

2 1 2 2 4 2 4 2 4 6 2 6

De anvendte divisorer fremkommer nu paa følgende maade.: Vi sætter først divisoren til nul og adderer det første tal, altsaa 2. Resultatet er den første divisor, 2. Derefter adderes næste tal, 1, og vi faar næste divisor, 3, o.s.v. De første tolv divisorer bliver da:

2 3 5 7 11 13 17 19 23 29 31 37

Disse divisorer er alle primtal, men nu er der ikke flere tal i vores række af tolv talkonstanter, og vi skal derfor begynde forfra, dog ikke fra det første 2-tal, men fra det første 4-tal, altsaa de otte sidste tal i konstantrækken:

4 2 4 2 4 6 2 6

Bruger vi disse tal til at addere videre med fra 37 faar vi:

41 43 47 49 53 59 61 67

Saa begynder vi forfra med 4-tallet og faar:

71 73 77 79 83 89 91 97

og saaledes videre. I de to sidste rækker af divisorer er der følgende ikke-primtal:

49 = 7×7
77 = 7×11
91 = 7×13

Grunden hertil er, at vores sigtesystem kun udelukker tal med primfaktorer 2, 3 og 5, ikke højere. Den grundlæggende række af differenser:

4 2 4 2 4 6 2 6

kan let konstrueres, hvis man tager den naturlige talrække og udskyder alle multipla af 2, 3 og 5. De tal, som bliver tilbage, vil da have disse differenser. Perioden i differenssystemet er 2×3×5 = 30.

De tre sidste celler: 26-28 bruges til lagring af værdien af de tre variable: test number, limit og divisor.

De 14 celler med maskinkodeordrer gennemgaas nu enkeltvis.

Celle 0. grn r+28. Nulstil divisorcellen. Adressen r+28 betyder cellen 28 pladser længere fremme (relativt til den celle, ordren staar i).

Celle 1. pa r+3 t +9. Sæt adressetallet 9 i celle 4.

Celle 2. pm a3, gm r+24. Adressen a3 refererer til adressen paa den tredje variable i navnelisten efter core code, altsaa her test number. Ordren pm a3 betyder: sæt M-registret lig med tallet med adressen a3, altsaa test number. Ordren gm r+24 gemmer M-registret 24 celler længere fremme, altsaa i celle 26. Resultatet er altsaa en flytning af test number fra den celle, hvor det staar i hovedprogrammet, via M-registret til vor lokale celle 26. Bemærk, at den lokale celle 26 her blot er nummereret ud fra den første celle (0) i maskinprogrammet.

Celle 3. pm a4, gm r+24. En analog flytning af den fjerde variable, limit, til celle 27.

Celle 4. arn r+0 t+1 IPA. Vi husker, at der i celle 1 blev indsat et 9-tal i adressen her. Ordren lyder derfor:

arn r+9 t +1 IPA

Naar denne ordre udføres, vil 1-tallet tilhøjre først blive adderet til 9-tallet:

arn r+10 t +1 IPA

Derefter udføres den del af ordren, der hedder arn r+10. Her opererer vi med maskinens resultatregister, R-registret. Først bevirker n-et, at R-registret nulstilles. Derefter adderes den celle, som ligger 10 pladser længere fremme til R-registret. Med andre ord, R-re-

gistret bliver sat lig med indholdet af celle 14, d.v.s. et 2-tal. Betegnelsen IPA betyder: indicer a-mærkning i bit PA af indikatoren. Af talkonstanterne i celle 14-25 har kun celle 25 et a-mærke, de andre ikke.

Celle 5. ac r+23, pmn r+21. Af disse to ordrer betyder ac r+23 addition af R-registret til cellen 23 pladser længere fremme, altsaa til celle 28. Da denne divisor-celle blev sat til nul i celle 0, er divisor nu lig med 2, den første divisor, vi skal prøve med. Den anden ordre, pmn r+21, nulstiller R-registret og sætter M-registret lig med indholdet af celle 5+21 = 26, altsaa testnumber.

Celle 6. pa r-2 t+13 LPA. Betegnelsen LPA betyder, at denne ordre kun skal udføres, hvis der er indiceret a-mærkning i PA, altsaa hvis vi efterhaanden er kommet frem til at addere det 6-tal, som staar i celle 25. Ordren vil da indsætte adressetallet 13 i cellen to pladser tilbage, altsaa i celle 4. Naar vi derefter kommer tilbage til celle 4 hedder den:

arn r+13 t +1 IPA

Først bliver 1-tallet adderet til 13 og r+14 giver 18. Divisortilvæksten flyttes altsaa tilbage fra 6-tallet i celle 25 til 4-tallet i celle 18.

Celle 7. arn r+20, sr r+21. Dette er ækvivalent med:

R := limit;
R := R - divisor;

altsaa bliver R lig med limit - divisor (sr betyder subtraktion).

Celle 8. hhn r+3 LT. Betegnelsen LT betyder, at denne ordre kun skal udføres, hvis R-registret er negativt, ellers springes den over. Selve ordren betyder: hop til højre halvcelle tre pladser længere fremme og nulstil R-registret. R bliver negativ, naar divisor bliver større end limit, altsaa naar undersøgelsen er afsluttet.

Celle 9. dln r+19 X. M-registret, der jo fik værdien af test number i celle 5, bliver nu divideret med celle 9+19 = 28, altsaa divisor. Efter divisionen findes kvotienten i R-registret og resten i M-registret, men da ordren er X-mærket, ombyttes R og M. Divisionsresten er derfor nu i R-registret.

Celle 10. hv r-6 NZ. Betingelsen NZ betyder, at ordren kun skal udføres, hvis $R \neq 0$, altsaa hvis divisionen ikke er gaaet op, ellers

overspringes ordren. Udføres ordren (hv r-6), hopper vi 6 pladser baglæns til celle 4. Der faar vi saa adderet en ny tilvækst til divisoren, idet der anden gang staar r+10+1, som giver 15, d.v.s. at 1-tallet i celle 15 bliver adderet til divisoren 2 med resultatet 3, o.s.v.

Celle 11. arn r-1, gr a2. Hvis divisionen er gaaet op, kommer vi frem til venstre halvdel af denne celle (arn r-1) og derefter højre (gr a2). Hvis vi derimod er hoppet hertil fra celle 8, fordi divisor er blevet for stor, kommer vi kun til højre halvcelle (gr a2). Ordren gr a2 bevirker, at R-registret bliver gemt i cellen for den variable: divisible (nr. 2 i navnelisten). Da et negativt R svarer til en sand boolean og R nul eller positivt til falsk, skal vi altsaa gøre R negativ i ordren arn r-1, hvorimod det direkte indhop til højre halvcelle fra celle 8 skal være med R nul eller positivt. Det sidste er allerede i orden, da vi nulstillede R ved hoppet fra celle 8. Ordren arn r-1 gør R negativ, fordi vi sætter R lig med indholdet af celle 10, som er negativ paa grund af adressen -6.

Celle 12. pm r+16, gm a5. Den aktuelle værdi af divisoren i celle 28 flyttes tilbage til hovedprogrammets divisorcelle.

Celle 13. hr s+1. Returhop til hovedprogrammet til cellen een plads længere fremme end derfra, hvor hoppet til maskinkoden blev gjort.

Naar man fra hovedprogrammet skal hoppe til maskinkoden, sker det med ALGOL-sætningen:

```
gier(code1);
```

Her skal code1 være navnet paa den første variable i navnelisten efter core code. Det skal være en simpel boolean i samme blok som maskinkoden. Oversætterprogrammet vil i code1 anbringe følgende ordre:

```
hv <celle 0> , qq n
```

hvori <celle 0> er adressen paa celle 0 i maskinkoden og n er antallet af celler i maskinkoden. Selve sætningen gier (code1) oversættes til:

```
hs <adressen paa code1>
```

altsaa et sekvenshop til cellen med code1.

Naar vi beregner primtal, ikke primfaktorer, har vi kun brug for at hoppe ind i celle 0 i maskinkoden. Men naar vi beregner primfaktorer, vil vi gerne hoppe ud af maskinkoden, naar en divisor er fundet, til trykning af faktoren for at hoppe ind i maskinkoden igen til fortsat division, uden at der kommer kludder i divisortilvæksterne. Det nye indhop skal ske i celle 2 for at undgaa nulstillingen af divisoren i celle 0 og retableringen af startadressen i celle 1.

For at kunne lave et indhop i celle 2, laver vi følgende stykke ekstra kode:

```
core code code2;
2, 46;
qq  0
e;
```

Dette stykke maskinkode indeholder kun den tomme celle (qq 0), men det vi har brug for, er blot den logiske variable, code2, som vi vil bruge i sætningen:

```
gier(code 2);
```

til indhop i celle 2. code 2 maa derfor beregnes som:

```
code2 := boolean(integer code1 + integer 10 2);
```

Dette dunkle udtryk kan udlægges saaledes: Vi starter med den logiske variable, code1, der indeholder:

```
hv <celle 0> , qq n
```

Vi skal blot øge adressen med 2, altsaa addere 2 i position 9. I GIER ALGOL 4 skrives et 2-tal i position 9 som 10 2, idet position 9 er den tiende bit i cellen. Nu er 10 2 definitionsmæssigt af typen boolean og for at kunne adderes, maa den fortolkes som en integer. Derfor skriver vi integer 10 2. Ogsaa code1 maa have denne betegnelse foran sig før vi kan addere:

```
integer code1 + integer 10 2
```

Resultatet af additionen skal imidlertid være af typen boolean, fordi code2 er af typen boolean. Altsaa maa vi skrive:

```
code2 := boolean (integer code1 + integer 10 2);
```

Bemærk, at den sidste sætning ikke hører med til den egentlige deklaration af core code code1 og core code code2. Det er en ganske almindelig ALGOL-sætning, som blot skal udføres, inden vi skriver gier (code2).

Bemærk ogsaa, at de to deklarationer af core code code1 og core code code2 begge afsluttes med et understreget e og semikolon:

```
e;
```

som tegn paa, at man forlader maskinkoden og vender tilbage til ALGOL.

4.2. Proceduren NEXT PRIME.

Denne procedure har deklarationen:

```
integer procedure NEXT PRIME(x);
integer x;
begin
   test number := x;
   divisible := true;
   for test number := test number + 2 while divisible do
   begin
      limit := sqrt(test number) + 0.5;
      gier(code1)
   end for test number;
   NEXT PRIME := x := test number - 2
end NEXT PRIME;
```

Til et opgivet heltal, x, der er en formel parameter, beregner proceduren det næste primtal i talrækken. Procedurefunktionen NEXT PRIME

antager selv værdien af det næste primtal, og det gør x ogsaa. Det er en forudsætning, at det oprindelige x er ulige.

Først sættes test number lig med x og divisible sættes til sand. Derefter kommer for-sætningen:

```
for test number := test number + 2 while divisible do
```

Da divisible først blev sat til sand, faar vi altid øget testnumber med 2. Indholdet af for-sætningen er blot de to sætninger:

```
limit := sqrt(test number) + 0.5;
gier(code1);
```

Vi beregner limit som kvadratroden af test number med en yderligere addition af 0.5 til resultatet. Vi har tidligere set, at den største divisor, som vi behøver at afprøve, er kvadratroden af det undersøgte tal. Naar der yderligere adderes 0.5, er det for at undgaa afrundingsfejl, som f. eks. ved testnumber = 49, hvor kvadratroden af 49 kunne risikere at blive 6.99999....

Sætningen gier(code1) bevirker hop til maskinprogrammets celle nul som forklaret ovenfor. Naar vi kommer tilbage fra maskinprogrammet, er divisible sat til falsk eller sand efter som test number var et primtal eller ej. Saalænge divisible er sand, kører for-sætningen videre med øgning af test number med to hver gang. Naar divisible bliver falsk, har vi fundet et primtal, og dette beregnes som test number minus 2, idet vi jo er kørt een gang for langt i for-sætningen.

Den opmærksomme læser vil nu spørge, hvorfor vi øger test number med 2 hver gang istedet for at bruge Eratosthenes si, som ville give færre gennemløb af for-sætningen og dermed færre beregninger af kvadratroden og færre kald af maskinkoden. Svaret er ligetil. Det, som tager lang tid ved primtalsberegningerne, er at vise, at et primtal virkeligt er et primtal, fordi der skal udføres et meget stort antal divisioner førend divisoren bliver større end kvadratroden af tallet. Hvis vi havde indrettet øgningen af test number til at bruge tilvæksterne: 4, 2, 4, 2 ..... o.s.v. i stedet for et konstant 2-tal, havde vi blot opnaaet at udelukke multipla af 3 og 5 i værdierne af test number. Men disse værdier bliver alligevel fanget øjeblikkeligt i den viste udformning, fordi de første divisorer er 2, 3 og 5. Det reelle tab af regnetid er derfor forsvindende.

Bemærk iøvrigt, at maskinkodens division med 2 er overflødig her, fordi test number altid er ulige. Det skal dog være i maskinkoden af hensyn til beregning af primfaktorer. Ogsaa her er tabet af regnetid forsvindende.

4.3. Proceduren PRINT FACTORS.

Deklaration herfor er:

```
procedure PRINT FACTORS(x);
value x;
integer x;
begin
   factor limit := limit := sqrt(x) + 1;
   first := true;
   test number := x;
   gier(code1);
   factor := divisor;
A: if divisible then
   begin
      power := 0;
      for power := power + 1 while test number:factor×factor = test number do
      test number := test number:factor;
      power := power - 1;
      if power > 1 then
      begin
         if ¬ first then writetext(‹×›);
         PRINT LEFT(factor, places);
         if power > 1 then
         begin
            writetext(‹^›);
            PRINT LEFT(power, places)
         end if power > 1;
         factor limit := limit := 1 + sqrt(test number);
         if first then first := power = 1 ∧ test number = 1
      end if power > 1;
      if factor ≤ factor limit then
      begin
         gier(code2);
         factor := divisor;
```

```
            go to A
         end if more;
      end if divisible;
      if first then
      begin
         prime count := prime count + 1;
         WRITE TEXT(
         {< Primtal}, {< Prime number},
         {< Nombre premier}, {< Primzahl})
      end if prime
      else
      if test number > 1 then
      begin
         writetext({<*});
         PRINT LEFT(test number, places)
      end if remaining prime factor
end PRINT FACTORS;
```

Proceduren bestemmer primfaktorerne i et opgivet heltal, x, der er en formel parameter i proceduren. De fundne primfaktorer trykkes, efterhaanden som de findes, med de mindste først. Er x et primtal, trykkes ordet: primtal.

Ligesom i NEXT PRIME begynder vi med at beregne kvadratroden af x:

```
factor limit := limit := sqrt(x) + 1;
```

Her er limit den grænse, som maskinkoden bruger, og factor limit er en anden grænse, som bruges af proceduren. Den logiske variable: first sættes til sand, test number sættes lig med x, og vi kalder maskinkoden med gier(code1). Efter returhoppet sætter vi factor lig med divisor, som er den sidst anvendte divisor.

Vi er nu naaet frem til etiketten A. Hvis divisible er sand, er factor aabenbart divisor i test number, og vi skal finde hvor mange gange, faktoren gaar op. Først sættes potensen, power, til nul, og vi udfører for-sætningen:

```
for power := power + 1 while
test number:factor*factor = test number do
test number := test number:factor;
```

For hvert gennemløb øges power med 1, og test number divideres med factor, saa længe divisionen gaar op. Da power ogsaa øges med 1, naar divisionen ikke mere gaar op, maa vi bagefter trække 1 fra power.

Nu skal faktoren trykkes. Som en ekstra sikkerhedsforanstaltning undersøger vi først om power ≥ 1. Hvis first er falsk, trykkes et gangetegn, idet der da har været trykt andre faktorer tidligere. Derefter trykkes værdien af factor med proceduren PRINT LEFT. Hvis power er større end 1, trykkes tegnet for potensopløftning (↑) og talværdien af power, ligeledes med PRINT LEFT.

Da test number nu er blevet mindre, beregner vi en ny værdi af limit og factor limit.

Da vi nu har trykt en primfaktor, skulle first egentlig sættes til falsk som tegn paa, at vi skal trykke et gangetegn foran eventuelle senere primfaktorer. Men da first lig sand ogsaa bruges til at indicere, at det oprindelige x har været et primtal, maa vi sikre os, at x ikke har været et primtal, hvor vi har bortdivideret primfaktoren. Hvis first er sand, skal first derfor forblive sand, hvis power er lig med 1 og test number er blevet divideret ned til 1.

Hvis den sidst brugte divisor, factor, er mindre end eller lig med den nu genudregnede factor limit, maa vi paa jagt efter nye divisorer. Det sker ved kald af maskinkoden med indhop i celle 2: gier(code2), som forklaret ovenfor. Efter returhoppet sættes factor lig med divisor, og vi gaar tilbage til etiketten A.

Til sidst i proceduren undersøger vi om first er sand. Er dette tilfældet, tælles der 1 frem i primtalstælleren, prime count, og vi trykker ordet Primtal. Hertil naar vi ogsaa, hvis det første kald af maskinkoden giver divisible falsk.

Hvis first er falsk, skal vi undersøge, om test number er større end 1. Hvis dette er tilfældet, findes den højeste primfaktor i tallet kun i potensen 1 og bliver derfor ikke bortdivideret. Altsaa maa vi trykke dens talværdi med et gangetegn foran.

Selv om proceduren PRINT FACTORS indeholder temmelig mange ALGOL-sætninger, foregaar de tidskrævende dele kun i maskinkoden og derfor med størst mulig hastighed.

5. PROGRAMMET

Selve-ALGOL-programmet til beregning af primtal er vist i afsnit 7.1, side 35. En typisk udskrift fra en kørsel er vist i afsnit 7.2 paa side 37.

Programmet indeholder kun eet blokniveau, og der er her deklareret følgende logiske variable:

first: Sand for primtal, falsk, naar en faktor er bortdivideret.

primes only: Sand, hvis vi kun skal bestemme primtal, ikke primfaktorer.

divisible: Sand, hvis tallet kan divideres.

code1: Indhopsvariabel til maskinkodens celle 0.

code2: Indhopsvariabel til celle 2 i maskinkoden.

Der er følgende heltalsvariable:

end: Det største tal, der skal undersøges.

factor: Den aktuelle værdi af en fundet primfaktor.

lang: Sprogtæller (1: Dansk, 2: Engelsk, 3: Fransk, 4: Tysk).

linerest: Antal frie linier tilbage paa en side.

modulus: Er lig med tallet 100000000000. Bruges ved venstrenormaliseret trykning af tal.

number: Den aktuelle værdi af det tal, der skal undersøges.

place rest: Det resterende antal anslagspladser paa en linie.

places: Antal udførte anslag ved trykning af et tal.

power: Primfaktorpotens.

prime count: Tælleværk for primtal.

series: Antallet af tal, som skal undersøges.

start: Det første tal, som skal undersøges.

limit: Maskinkodens øvre grænse for de anvendte divisorer.

test number: Det aktuelle tal, som divideres af maskinkoden.

factor limit: Divisorgrænse under primfaktorbestemmelsen.

divisor: Den anvendte divisor.

De anvendte proceduredeklarationer er vist i de tidligere afsnit og er ikke medtaget i programudskriften.

Programmet begynder med at indsætte startværdien 69 af linerest og at tildele modulus sin værdi. Derefter kaldes:

```
select(17);
```

som indsætter skrivemaskinen som input- og outputmedium. Saa vælges sproget med SELECT LANGUAGE, og programmet udskriver tekstlinien:

PROGRAM DEMON-7. Beregning af primtal og primfaktorer.

Med proceduren ASK NUMBER spørger maskinen nu om værdien af det første tal, som skal undersøges. Dette tal gemmes som heltalsvariablen, start.

Bemærk, at der før kaldet af ASK NUMBER er kodet:

```
char := 0;
```

char indeholder den sidst indlæste karakter og bør altid nulstilles først i et program.

Programmet undersøger nu, om der er gjort forsøg paa at drille ved at opgive et negativt tal som startværdi. Er dette tilfældet, sættes start til 2 uden nærmere kommentar.

Derefter spørger maskinen om hvor mange tal, der skal undersøges. Det opgivne tal gemmes i den variable: series.

Med proceduren QUESTION spørger programmet nu, om vi blot skal bestemme primtal. Er svaret ja, sættes primes only til sand, og svares der nej, sættes den til falsk, og vi faar beregning af primfaktorer.

Nu nulstilles primtalstælleren, og antallet af anslag paa linien sættes til 65. Det største tal, der skal undersøges, beregnes som:

```
end := start + series - 1;
```

Da denne addition kan give spild, nemlig hvis grænsen for en integer overskrides (549755813887), vil end blive forkert i dette tilfælde. Det kørende ALGOL-program kontrollerer ikke om der forekommer spild ved heltalsaddition, men det er let at se bagefter, idet end da er blevet negativ. Er dette sket, sætter maskinen end lig med 100. Hvis start er mindre end 2, sættes den lig med 2.

Skal der kun bestemmes primtal, sættes number først lig med start. Er number = 2, trykker vi dette 2-tal med PRINT PRIME(2) og sætter number lig med 3. Derefter korrigeres number nedad til det nærmeste lavere ulige tal:

```
number := number - (if number:2×2 = number then 1 else 2);
```

og vi kalder den centrale for-sætning til primtalsberegningen:

```
for number := NEXT PRIME(number)
while number ≤ end do
PRINT PRIME(number);
```

Her bliver number hele tiden øget til det næste højere primtal, og det trykkes, saa længe det er mindre end eller lig med end. Bemærk iøvrigt, at det sidste primtal, der beregnes, altid vil være større end slutværdien, end, men det bliver ikke trykt. Dette kan være noget generende, hvis det er et meget stort primtal, for hvilket regnetiden er nogle minutter.

Ved beregningen af primfaktorer har vi en simpel for-sætning:

```
for number := start step 1 until end do
```

For hver værdi af number trykkes en ny linie, værdien af number, 3 mellemrum og endelig primfaktorerne med PRINT FACTORS.

Som afslutning paa beregningen trykker programmet antallet af fundne primtal, samt værdien af:

```
end/ln(end) - start/ln(start)
```

som er en tilnærmet værdi af det antal primtal, man kunne forvente i intervallet. Af beregningseksemplet side 38 ser man, at der naturligvis er en betydelig forskel paa det faktiske antal primtal og det forventede, navnlig for smaa intervaller.

Til slut spørger programmet, om man ønsker at regne igen, og er det tilfældet, hoppes tilbage til etiketten A.

6. REFERENCER

Beiler, A. H.: Recreations in the Theory of Numbers
- The Queen of Mathematics Entertains.
Dover, 1964.

Kjær, J.: Tændstikspillet NIM, et GIER
Demonstrationsprogram, Haldor Topsøe, 1966.

7. APPENDIKS

7.1. ALGOL-programmet

```
Program DEMON-7. Prime numbers.
begin
   boolean first, primes only, divisible, code1, code2;
   integer end, factor, lang, linerest, modulus, number, place rest, places,
   power, prime count, series, start, limit, test number, factor limit,
   divisor;
   comment Her anbringes proceduredeklarationerne, f. eks. med comment library
eller med copy;
   linerest := 69;
   modulus := 100000000000;
   select(17);
   SELECT LANGUAGE;
   LINE;
   WRITE TEXT(
   {<PROGRAM DEMON-7. Beregning af primtal og primfaktorer.},
   {<PROGRAM DEMON-7. Calculation of prime numbers and prime factors.},
   {<PROGRAMME DEMON-7. Calcul des nombres premiers et des facteurs premiers.},
   {<PROGRAMM DEMON-7. Berechnung von Primzahlen und Primfaktoren.});
A: LINE;
   char := 0;
   start := ASK NUMBER(
   {<Opgiv det første tal, der skal undersøges},
   {<Specify the first number to be tested},
   {<Specifiez le premier nombre a examiner},
   {<Bitte, die erste Zahl der Untersuchung angeben});
   if start < 0 then start := 2;
   series := ASK NUMBER(
   {<Hvor mange tal skal undersøges},
   {<How many numbers should be tested},
   {<Combien de nombres faut-il examiner},
   {<Wieviel Zahlen sollen untersucht werden});
```

```
primes only := QUESTION(
{<Skal vi blot bestemme primtal},
{<Shall we only find the prime numbers},
{<Faut-il trouver seulement les nombres premiers},
{<Sollen wir nur die Primzahlen finden});
LINE;
LINE;
prime count := 0;
place rest := 65;
end := start + series - 1;
if start < 2 then start := 2;
if end < 0 then end := 100;
if primes only then
begin
   number := start;
   if number = 2 then
   begin
      PRINT PRIME(2);
      number := 3
   end if 2;
   number := number - (if number÷2×2 = number then 1 else 2);
   for number := NEXT PRIME(number) while number ≤ end do
   PRINT PRIME(number)
end if primes only
else
begin
   for number := start step 1 until end do
   begin
      LINE;
      PRINT LEFT(number, places);
      writetext({<   });
      PRINT FACTORS(number)
   end for number
end print factors;
LINE;
LINE;
```

```
    WRITE TEXT(
    {<Antal primtal                 },
    {<Number of primes              },
    {<Nombre de nombres premiers    },
    {<Anzahl von Primzahlen         });
    write({-ddddddddddddd}, prime count);
    LINE;
    writetext({<delta(N/ln(N))                  });
    write({-ddddddddddddd}, end/ln(end) - start/ln(start));
    LINE;
    if QUESTION(
    {<Ønsker De at regne igen},
    {<Do you want to calculate more},
    {<Voulez-vous calculer encore},
    {<Wollen sie nochmals rechnen}) then go to A
end program;
```

7.2. Udskrift fra en kørsel.

```
Select language: d: danish, e: english, f: french, g: german.: d
Dansk

PROGRAM DEMON-7.  Beregning af primtal og primfaktorer.

Opgiv det første tal, der skal undersøges: 1234567,
Hvor mange tal skal undersøges: 100,
Skal vi blot bestemme primtal: ja

 1234577 1234603 1234613 1234627 1234657

Antal primtal                    5
delta(N/ln(N))                   7

Ønsker De at regne igen: ja
```

Opgiv det første tal, der skal undersøges: 123456789,
Hvor mange tal skal undersøges: 23,
Skal vi blot bestemme primtal: nej

123456789	$3\uparrow 2\times 3607\times 3803$
123456790	$2\times 5\times 37\times 333667$
123456791	Primtal
123456792	$2\uparrow 3\times 3\times 59\times 87187$
123456793	157×786349
123456794	$2\times 19\times 113\times 28751$
123456795	$3\times 5\times 7\times 11\times 89\times 1201$
123456796	$2\uparrow 2\times 30864199$
123456797	73×1691189
123456798	$2\times 3\uparrow 4\times 769\times 991$
123456799	29×4257131
123456800	$2\uparrow 5\times 5\uparrow 2\times 154321$
123456801	$3\times 13\times 23\times 137633$
123456802	$2\times 7\times 61\times 144563$
123456803	Primtal
123456804	$2\uparrow 2\times 3\times 10288067$
123456805	$5\times 17\times 1452433$
123456806	$2\times 11\times 547\times 10259$
123456807	$3\uparrow 2\times 13717423$
123456808	$2\uparrow 3\times 353\times 43717$
123456809	7×17636687
123456810	$2\times 3\times 5\times 1049\times 3923$
123456811	Primtal

Antal primtal	3
delta(N/ln(N))	1

Ønsker De at regne igen: nej.

↑

?