La conjecture se résout par récurrence avec un raisonnement par l'absurde, en utilisant la propriété du fonctionnement de l'algorithme de Goldbach dans les congruences. Quelque soit une limite n = 15k criblée ⇒ la vérification de la décomposition de 2n =30k = p+q , cela implique la vérification de la conjecture sur plusieurs limite n = 15(k+1,+2...+x) successives modulo 15.

Du fait de la propriété récurrente de l'algorithme suite au décalage d'un rang des congruences sur leur successeur, cela implique que: en même temps qu'une limite 2n est vérifiée, elle est aussi vérifiée pour plusieurs limite n successives modulo 15, suite à ce décalage d'un rang des congruences. Démontré dans un document à cet effet.

Nombre de couples P+q = 2n

Pour limite n de 21 000 000 007 à 21 000 000 127 en progression arithmétique modulo 15

Nombre d'entiers G(n), avec A ≠ 2n [P] («non congrus») Fam 7. **Crible G**

===== RESTART: E:\Documents\Conjecture de Goldbach\Crible_G.T.Y_modulo30.py ====

Donnez N: 21000000067

Nombres ***A(n)*** non congrus à 2n[P] 7 à 21000000067 famille 7 = premiers de 21000000067 à 42000000134: **108 681 829**

Donnez N: 21000000061 pour la Fam 1.

Nombres ***A(n)*** non congrus à 2n[P] 1 à 21000000061 famille 1 = premiers de 21000000061 à 42000000122: **108 685 771**

Donnez N: 36000000071

Nombres ***A(n)*** non congru 2n[P] 1 à 36000000071 famille 11 premiers de 36000000326 à 72000000142: 182 249 978 -----

Formule d'Estimation du nombre de couples P+q = 2n Par famille :

1_) On calcule directement avec la formule du TNP, qui en est une conséquence directe, le nombre de nombres premiers **A = p** appartenant à ***A(n)*** l'ensemble des entiers ***A non congrus à 2n [P]*** avec ***P ≤√ 2n ;*** *par Famille, soit le nombre de nombres premiers* ***q ∈ [n ; 2n] qui vaut environ G(n) conséquence directe du TNP.***

$$G(n) = \frac{n}{\ln(2n)}$$

2_) Soit environ : **108681829 / Ln 108681829 = 5873444 et 5873444 * 1,323323 = 7 754 843 couples P+q = 2n.**

Qui sera donc, inférieur au nombre réel de couples p+q qui décomposent 2n en somme de deux nombres premiers.

Ce nombre de couples serra oscillatoire + ou - , lorsque la limite n + 15 tend vers l'infini.

La variation en plus ou en moins se situe entre l'estimation et le nombre réel de couples. Le nombre de solutions qui vérifie un entier 2n est un effet boule de neige ! Lorsque 2n tend vers l'infini .

Vérification par Fam(i) :

Ecrible : Réel fam (1) : Nombre d'entiers A = P' ≤ n = 6 000 000 150 ; 34 942 490

n = 6000000150 ; fam 1 vaut ~ (G(n) / 8) ~ 32 316 203,20 entiers A ≢ 2n[P]

Gcrible : Réel fam (1) : Nombre d'entiers A ≢ 2n[P] ≤ n = 6 000 000 150 ; où G(n) = 32 748 571
32748571 A ou Nombres premiers q à plus ou moins 1

la fonction G(2n) vaut ~ : $C_2 * \frac{G(n)}{\ln G(n)}$ = *1,320323 * 1892502,95 = 2498715,17 couples*

p+q = 2n
ou encore: 1892502,95 * 2 = 3785005,89

Python 3.9.0+ (default, Oct 20 2020, 08:43:38)
[GCC 9.3.0] on linux

>>> **nombre de couple Réels:**
========== RESTART: /home/gilbert/Programmes/Crible_EG2_mod30.py ==========
Donnez n: 6000000000
Nombres P' non congruent à [pi] de 1 à 6000000000 famille 1, nombr de couples p+q=2n: **5 372 389** ----- 50.46 secondes
>>>
Donnez n: 6000000015
Nombres P' non congruent à [pi] de 1 à 6000000015 famille 1, nombr de couples p+q=2n: **5 371 836** ----- 50.19
>>>
Donnez n: 6000000030
Nombres P' non congruent à [pi] de 1 à 6000000030 famille 1, nombr de couples p+q=2n: **5 371 282** ----- 50.78 **minimum**
>>>
Donnez n: 6000000045
Nombres P' non congruent à [pi] de 1 à 6000000045 famille 1, nombr de couples p+q=2n: **5 864 795** ----- 51.06
>>>
Donnez n: 6000000060
Nombres P' non congruent à [pi] de 1 à 6000000060 famille 1, nombr de couples p+q=2n: **5 726 738** ----- 50.5
>>>
Donnez n: 6000000075
Nombres P' non congruent à [pi] de 1 à 6000000075 famille 1, nombr de couples p+q=2n: **5 369 565** ----- 50.37
>>>
Donnez n: 6000000090
Nombres P' non congruent à [pi] de 1 à 6000000090 famille 1, nombr de couples p+q=2n: **6 699 532** ----- 50.37 **maximum**
>>>
Donnez n: 6000000105
Nombres P' non congruent à [pi] de 1 à 6000000105 famille 1, nombr de couples p+q=2n: **6 135 775** ----- 51.53
>>>
Donnez n: 6000000120
Nombres P' non congruent à [pi] de 1 à 6000000120 famille 1, nombr de couples p+q=2n: **5 699 221** ----- 50.18
>>>
Donnez n: 6000000135
Nombres P' non congruent à [pi] de 1 à 6000000135 famille 1, nombr de couples p+q=2n: **5 371 467** ----- 50.77
>>>
Donnez n: 6000000150
Nombres P' non congruent à [pi] de 1 à 6000000150 famille 1, nombr de couples p+q=2n: **5 474 722** ----- 50.79
>>>
Donnez n: 6000000165
Nombres P' non congruent à [pi] de 1 à 6000000165 famille 1, nombr de couples p+q=2n: **5 386 483** ----- 52.14
>>>
Donnez n: 6000000180
Nombres P' non congruent à [pi] de 1 à 6000000180 famille 1, nombr de couples p+q=2n: **5 379 222** ----- 50.73
>>>

Nombres P' non congruent à [pi] de 1 à 6000000195 famille 1, nombr de couples p+q=2n: **6 540 191** ----- 50.35
>>>
Donnez n: 6000000210
Nombres P' non congruent à [pi] de 1 à 6000000210 famille 1, nombr de couples p+q=2n: **5 373 948** ----- 53.29
>>>
Donnez n: 6000000225
Nombres P' non congruent à [pi] de 1 à 6000000225 famille 1, nombr de couples p+q=2n: **5 626 260** ----- 51.86
>>>
Donnez n: 6000000240
Nombres P' non congruent à [pi] de 1 à 6000000240 famille 1, nombr de couples p+q=2n: **6 076 867** ----- 51.48
>>>
Donnez n: 6000000255
Nombres P' non congruent à [pi] de 1 à 6000000255 famille 1, nombr de couples p+q=2n: **5 373 598** ----- 50.86
>>>
Donnez n: 6000000270
Nombres P' non congruent à [pi] de 1 à 6000000270 famille 1, nombr de couples p+q=2n: **6 269 874** ----- 50.84
>>>
Donnez n: 6000000285
Nombres P' non congruent à [pi] de 1 à 6000000285 famille 1, nombr de couples p+q=2n: **5 382 058** ----- 50.13
>>>
Donnez n: 6000000300
Nombres P' non congruent à [pi] de 1 à 6000000300 famille 1, nombr de couples p+q=2n: **6 442 697** ----- 50.54
>>>

Fam 7 : *note le programme va afficher modulo [pi] au lieu de* [***P]*** *tel que défini ci dessus.*

*Érathothène = 115 541 853 P' Goldbach = 108 684 761 q ; différence : 6857092 * log 12 =**17 039 233***

===== RESTART: E:\Documents\Conjecture de Goldbach**Crible_ EG_ 2N_mod30.py** =====

Donnez N: 21000000007

Nombres P' non congruents **[P]** de 1 à 21000000007, ou couples p+q = 2n, **famille 7** : **20206074** ----- 120.73

Donnez N: 21000000022

Nombres P' non congruents [P] de 1 à 21000000022, ou couples p+q = 2n, **famille 7** : **16847927** ----- 120.99

Donnez N: 21000000037

Nombres P' non congruents [P] de 1 à 21000000037, ou couples p+q = 2n, famille 7 : **16970789** ----- 121.32

Donnez N: 21000000052

Nombres P' non congruents [P] de 1 à 21000000052, ou couples p+q = 2n, famille 7 : **16840036** ----- 122.85

Donnez N: 21000000067

Nombres P' non congruents [P] de 1 à 21000000067, ou couples p+q = 2n, famille 7 : **17701458** ----- 120.31

Donnez N: 21000000082

Nombres P' non congruents [P] de 1 à 21000000082, ou couples p+q = 2n, famille 7 : **17080949** ----- 121.09

Donnez N: 21000000097

Nombres P' non congruents [P] de 1 à 21000000097, ou couples p+q = 2n, famille 7 : **17830790** ----- 121.35

Donnez N: 21000000112

Nombres P' non congruents [P] de 1 à 21000000112, ou couples p+q = 2n, famille 7 : **22037139** ----- 121.53

Donnez N: 21000000127

Nombres P' non congruents [P] de 1 à 21000000127, ou couples p+q = 2n, **famille 7** : **17323994** ----- 121.54

Fam 1 : *Érathothène = 115 537 277 P' Goldbach = 108 685 478 q ; différence : 6851799 * log 12 =* ***17 026 080***

===== RESTART: E:\Documents\Conjecture de Goldbach\Crible_ EG_ 2N_mod30.py =====

Donnez N: 21000000001

Nombres P' non congruents [P] de 1 à 21000000001, ou couples p+q = 2n, **famille 1** : **16 995 045** ----- 117.26

Donnez N: 21000000016

Nombres P' non congruents [P] de 1 à 21000000016, ou couples p+q = 2n, **famille 1** : **17317580** ----- 119.12

Donnez N: 21000000031

Nombres P' non congruents [P] de 1 à 21000000031, ou couples p+q = 2n, famille 1 : **16839272** ----- 119.73

Donnez N: 21000000046

Nombres P' non congruents [P] de 1 à 21000000046, ou couples p+q = 2n, famille 1 : **16835203** ----- 120.29

Donnez N: 21000000061

Nombres P' non congruents [P] de 1 à 21000000061, ou couples p+q = 2n, famille 1 : **16835103** ----- 121.93

Donnez N: 21000000076

Nombres P' non congruents [P] de 1 à 21000000076, ou couples p+q = 2n, famille 1 : **18835406** ----- 119.78

Donnez N: 21000000091

Nombres P' non congruents [P] de 1 à 21000000091, ou couples p+q = 2n, famille 1 : **21551387** ----- 120.93

Donnez N: 21000000106

Nombres P' non congruents [P] de 1 à 21000000106, ou couples p+q = 2n, famille 1 : **16836460** ----- 119.87

Donnez N: 21000000121

Nombres P' non congruents [P] de 1 à 21000000121, ou couples p+q = 2n, famille 1 : **16838019** ----- 125.04

Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

Érathothène = 193 470 325 P' Goldbach = 182 244 035 q ; différence : ***11 226 290*** ** log 12 =* ***27 896 282***

Donnez N: 36000000015

Nombres P' non congruents [P] de 1 à 36000000015, ou couples p+q = 2n, **famille 11** : **33104786** ----- 209.17

Donnez N: 36000000030

Nombres P' non congruents [P] de 1 à 36000000030, ou couples p+q = 2n, famille 11 : **27612531** ----- 210.23

Donnez N: 36000000045

Nombres P' non congruents [P] de 1 à 36000000045, ou couples p+q = 2n, famille 11 : **27690511** ----- 205.71

Donnez N: 36000000060

Nombres P' non congruents [P] de 1 à 36000000060, ou couples p+q = 2n, famille 11 : **27578581** ----- 208.62

Donnez N: 36000000075

Nombres P' non congruents [P] de 1 à 36000000075, ou couples p+q = 2n, famille 11 : **32016762** ----- 210.74

Donnez N: 36000000090

Nombres P' non congruents [P] de 1 à 36000000090, ou couples p+q = 2n, famille 11 : **27586734** ----- 209.17

Donnez N: 36000000105

Nombres P' non congruents [P] de 1 à 36000000105, ou couples p+q = 2n, famille 11 : **30430001** ----- 211.37

Donnez N: 36000000120

Nombres P' non congruents [P] de 1 à 36000000120, ou couples p+q = 2n, famille 11 : **33093328** ----- 208.1

Donnez N: 36000000135

Nombres P' non congruents [P] de 1 à 36000000135, ou couples p+q = 2n, famille 11 : **31780760** ----- 215.35

Donnez N: 36000000150

Nombres P' non congruents [P] de 1 à 36000000150, ou couples p+q = 2n, famille 11 : 29850469 ----- 209.82

Donnez N: 36000000165

Nombres P' non congruents [P] de 1 à 36000000165, ou couples p+q = 2n, famille 11 : 27578646 ----- 208.65

Donnez N: 36000000180

Nombres P' non congruents [P] de 1 à 36000000180, ou couples p+q = 2n, famille 11 : 27576296 ----- 211.6

Donnez N: 36000000195

Nombres P' non congruents [P] de 1 à 36000000195, ou couples p+q = 2n, famille 11 : 28221298 ----- 210.93

Donnez N: 36000000210

Nombres P' non congruents [pi] de 1 à 36000000210, ou couples p+q = 2n, famille 11 : 27597897 ----- 215.65

Donnez N: 36000000225

Nombres P' non congruents [P] de 1 à 36000000225, ou couples p+q = 2n, famille 11 : **33 120 533** ----- 211.23

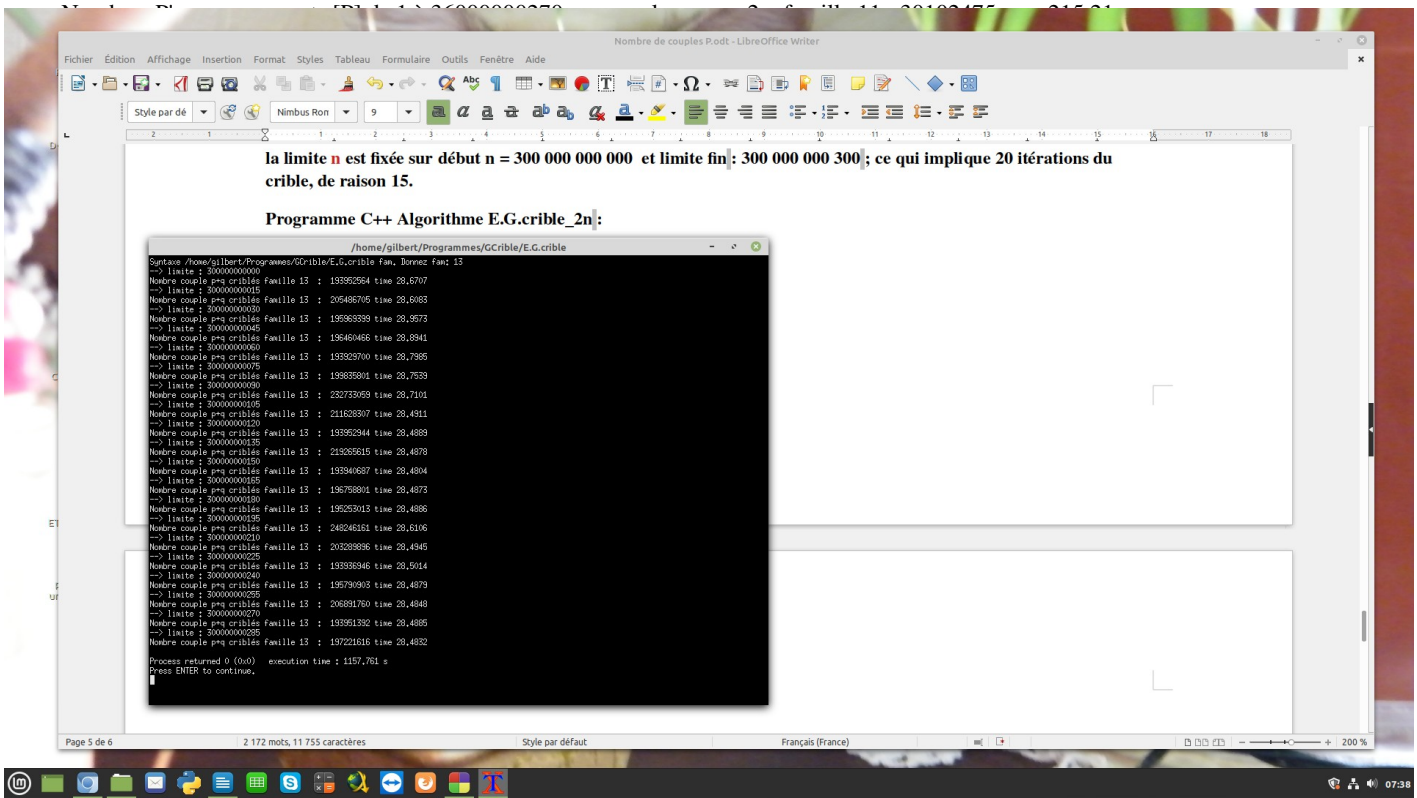
Donnez N: 36000000240

Nombres P' non congruents [P] de 1 à 36000000240, ou couples p+q = 2n, famille 11 : 27577908 ----- 209.84

Donnez N: 36000000255

Nombres P' non congruents [P] de 1 à 36000000255, ou couples p+q = 2n, famille 11 : 27580038 ----- 208.81

Donnez N: 36000000270



on utilisera n de la forme 15k ; n progressera modulo 15 entre le début et la fin , que l'on peut modifier, ligne code : 156, 157.

On peut aussi activer la // **ligne de code 106 ;** qui donne le nombre de nombre premiers P' criblé par la fonction **ECrible** Ératosthène dans le même temps, mais cela rallongera la durée du test et prend de l'espace mémoire en plus...

la limite n est fixée sur début n = 300 000 000 000 et limite fin : 300 000 000 300 ; ce qui implique 20 itérations du crible, de raison 15.

Programme C++ Algorithme E.G.crible_2n :

code source :

```
// -*- compile-command: "/usr/bin/g++ -g goldbachs.cc" -*-

#include <vector>

#include <iostream>

#include <cmath>

#include <stdlib.h>

#include <time.h>

using namespace std;

// fill Erathosthene sieve crible for searching primes up to 2*crible.size()*32+1

// crible is a (packed) bit array, crible[i] is true if 2*i+1 is a prime

// crible must be set to true at startup

void fill_crible(vector<unsigned> &crible, unsigned p)

{

 crible.resize((p - 1) / 64 + 1);

 unsigned cs = crible.size();

 unsigned lastnum = 64 * cs;

 unsigned lastsieve = int(std::sqrt(double(lastnum)));

 unsigned primesieved = 1;

 crible[0] = 0xfffffffe; // 1 is not prime and not sieved (2 is not sieved)

 for (unsigned i = 1; i < cs; ++i)

  crible[i] = 0xffffffff;

 for (; primesieved <= lastsieve; primesieved += 2)

 {

  // find next prime

  unsigned pos = primesieved / 2;

  for (; pos < cs; pos++)
```

```
    {
      if (crible[pos / 32] & (1 << (pos % 32)))
        break;
    }
    // set mutiples of (2*pos+1) to false
    primesieved = 2 * pos + 1;
    unsigned n = 3 * primesieved;
    for (; n < lastnum; n += 2 * primesieved)
    {
      pos = (n - 1) / 2;
      crible[(pos / 32)] &= ~(1 << (pos % 32));
    }
  }
}
unsigned nextprime(vector<unsigned> &crible, unsigned p)
{
  // assumes crible has been filled
  ++p;
  if (p % 2 == 0)
    ++p;
  unsigned pos = (p - 1) / 2, cs = crible.size() * 32;
  if (2 * cs + 1 <= p)
    return -1;
  for (; pos < cs; ++pos)
  {
    if (crible[pos / 32] & (1 << (pos % 32)))
    {
      pos = 2 * pos + 1;
      // if (pos!=nextprime(int(p)).val) CERR << "error " << p << endl;
      return pos;
    }
  }
  return -1;
}

typedef unsigned long long ulonglong;
```

```
size_t ECrible(const vector<ulonglong> &premiers, ulonglong n, int fam, vector<bool> &crible, size_t lencrible)
{ //on va contruire un tableau de 1 modulo 30 et rappeler les premiers p
 int cl = clock();
 // size_t lencrible = n / 30,
 size_t nbpremiers = premiers.size(); //on va contruire un tableau de 1 modulo 30 en divisant N par 30
 //vector<bool> crible(lencrible, true);  // on rappelle les nombres premiers p d'Eratotene ci dessus
 // ulonglong n2=2*n;
 vector<ulonglong> indices(nbpremiers);
 for (size_t i = 0; i < nbpremiers; ++i)
 {
  ulonglong p = premiers[i];
  ulonglong produit;
  int GM[] = {7, 11, 13, 17, 19, 23, 29, 31}; // on va calculer le produit de p par un element du groupe GM
  for (size_t j = 0; j < sizeof(GM) / sizeof(int); j++)
  {
   produit = p * GM[j]; // calcul du produit, jusqu'a ce que le produit soit égale à fam modulo 30
   if (produit % 30 == fam)
   {
    produit /= 30; // puis on va va calculer l'indice, afin de commencer à cribler de l'indice à n/30 et on réitère
    break;
   }
  }
  indices[i] = produit;
 }
 ulonglong nslices = lencrible / 1500000, currentslice = 0;
 if (nslices == 0)
  nslices = 1;
 for (; currentslice < nslices; ++currentslice)
 {
  size_t slicelimit = currentslice + 1;
  slicelimit = slicelimit == nslices ? lencrible : (currentslice + 1) * (lencrible / nslices);
  for (size_t i = 0; i < nbpremiers; ++i)
  {
   ulonglong p = premiers[i];
   size_t index;
```

```
    for (index = indices[i]; index < slicelimit; index += p)
      crible[index] = 0;
    indices[i] = index;
   }
  }
  size_t total = 0;
  for (size_t index = 0; index < lencrible; ++index)
   total += int(crible[index]);
  //cout << "Nombre premiers criblés famille " << fam << " plus petits que " << n << ": " << total << " time " << (clock() - cl) * 1e-6 << endl;
  return total; // à la fin du crible on return le résultat est le temps mis
}

size_t GCrible(const vector<ulonglong> &premiers, ulonglong n, int fam, vector<bool> &crible, size_t lencrible)
{
  int cl = clock();
  //size_t lencrible = n / 30,
  size_t nbpremiers = premiers.size(); //on utilise le tableau de 1 mod 30 criblé par Ératosthène ci dessus
  //vector<bool> crible(lencrible, true);  // on rappelle les nombres premiers p d'Eratotene ci dessus
  ulonglong n2 = 2 * n;
  vector<ulonglong> indices(nbpremiers);
  for (size_t i = 0; i < nbpremiers; ++i)
  {
   ulonglong p = premiers[i];
   ulonglong reste = n2 % p; // on calcule le reste de 2n par p
   if (reste % 2 == 0)
    reste += p;
   ulonglong pi2 = 2 * p;
   while (reste % 30 != fam) // tant que le reste += p n'est pas = à Fam % 30 on rajoute 2*p
    reste += pi2;
   reste /= 30; // on ensuite on va calculer l'indice pour commencer à cribler le tableau de 1.1.1.... avec p, de l'indice à n/30
   indices[i] = reste;
  }
  ulonglong nslices = lencrible / 1500000, currentslice = 0;
  if (nslices == 0)
```

```
  nslices = 1;
 for (; currentslice < nslices; ++currentslice)
 {
  size_t slicelimit = currentslice + 1;
  slicelimit = slicelimit == nslices ? lencrible : (currentslice + 1) * (lencrible / nslices);
  for (size_t i = 0; i < nbpremiers; ++i)
  {
   ulonglong p = premiers[i];
   size_t index;
   for (index = indices[i]; index < slicelimit; index += p)
    crible[index] = 0;
   indices[i] = index;
  }
 }
 size_t total = 0;
 for (size_t index = 0; index < lencrible; ++index)
  total += int(crible[index]); // le criblage du tableau de 1 modulo 30 jusqu'a n/30 (1.1.1.1...etc) est fini on va retourner le résultat
 cout << "Nombre couple p+q criblés famille " << fam <<  "  :  " << total <<  " time " << (clock() - cl) * 1e-6 << endl;
 return total;
}

int main(int argc, char **argv)
{
 vector<unsigned> temp;
 ulonglong debut = 300000000000;
 ulonglong fin = 300000000300;

 int fam = 1;
 if (argc > 1)
 {
  fam = atoi(argv[1]);
 }
 else
 {
  cout << "Syntaxe " << argv[0] << " fam. Donnez fam: ";
```

```
    cin >> fam;
  }

  for(ulonglong limite = debut; limite<fin; limite += 15){
    cout << "--> limite : " << limite << endl;
    double sqrt2N = unsigned(std::sqrt(2 * double(limite)));
    fill_crible(temp, sqrt2N);
    vector<ulonglong> premiers;
    for (ulonglong p = 7; p <= sqrt2N;)
    {
      premiers.push_back(p);
      p = nextprime(temp, p);
      if (p == unsigned(-1))
        break;
    }

    size_t lencrible = limite/30;
    vector<bool> crible(lencrible, true);
    ECrible(premiers, limite, fam, crible, lencrible);
    GCrible(premiers, limite, fam, crible, lencrible);
  }
}
```