Université de Limoges

Projet d'initiation à la recherche
Master 1 - Mathématiques
2016-2017

Groupe :
Bedouhene Abderahmane
Daoudi Sonia
Mazlani Kaoutar

Encadrants :
Clavier Christophe

Thème : Borne inferieure d'un éventuel nombre parfait impair

Sommaire

Introduction

Les mathématiques, l'une des plus fines expressions de l'esprit humain, elles ont été développées dans différentes cultures afin d'identifier ordre et cohérence, dans l'espoir de comprendre, prédire et peut-être de contrôler des évènements. Elles sont nées quand l'homme primitif sentit le besoin de compter et elles ont évolué pendant plus de quatre mille ans. Parmi les grandes branches des mathématiques, il y a l'arithmétique, comme son nom l'indique, *arithmos*, est l'étude élémentaire des nombres entiers, rationnels et de leur propriété ; Et de là naquirent plusieurs types de nombres, ils sont à l'image des organismes vivants, parmi ces nombres, il y a les singuliers, les sociables, les premiers, les amicaux, les parfaits, les automorphes, les intouchables, de Fermat, de Mersenne…etc.

Des nombres au propriétés curieuses, qui élargissent les connaissances de l'homme, c'est ainsi qu'en utilisant un nombre de Mersenne i.e. un nombre s'écrivant sous la forme d'une puissance première de 2 moins une unité, *Euclide* discernât les nombres parfaits III-siècle av. J.-C.

Ce nombre qui est entre l'abondance et la déficience, qui n'a ni trop peu, ni trop de diviseurs stricts, juste ce qu'il faut, de telle manière à ce que leur somme soit égale à ce même nombre dit parfait.

Et malgré l'ancienneté de leur découverte, ils sont à nos jours, l'objet de recherche, notamment celle de l'existence des nombres parfaits impairs, car à ce jour, les seuls nombres parfaits trouvés sont pairs et la question de leur existence reste en suspens, jusqu'à présent, les scientifiques n'ont été capable que de démontrer leur inexistence jusqu'à certaines bornes définies par les moyens de calcul dont ils disposent.

Dans ce projet, nous essayerons de voir plus en détails quels sont donc ces nombres à l'équilibre idéal et essayerons de démontrer qu'il n'existe pas de nombre parfaits impairs inferieurs à 10^{160}.

I. Présentation générale des nombres parfaits :

I. Qu'est-ce qu'un nombre parfait ?

I.1 Définition :

Soit N un entier naturel strictement positif, on désigne par σ(N) la somme des diviseurs de N ; N est dit parfait lorsque 2N = σ(N), autrement dit σ(N) – N = N. On peut donc dire : N est parfait si et seulement si N est égal à la somme de ses diviseurs stricts.

I.2 Exemple :

- Pour N = 5 : les diviseurs de 5 sont {1,5}, leur somme σ(N) = 1 + 5 = 6 ≠ 2 x 5. Le nombre 5 n'est donc pas parfait.
- Pour N = 6 : les diviseurs de 6 sont {1,2,3,6}, leur somme σ(N) = 1 + 2 + 3 + 6 = 12 = 2 x 6.
  Le nombre 6 est donc parfait.

I.3 Abondance d'un nombre parfait :

L'abondance d'un nombre N est le rapport de la somme de ses diviseurs sur lui-même, étant donné 2N = σ (N) alors l'abondance d'un nombre parfait $\frac{\sigma(N)}{N} = 2$.

II. Les nombres parfaits pairs :

II.1 Diverses définitions et propriétés :

- Euclide remarqua que les nombres parfaits s'écrivent sous la forme suivante :

$$6 = 2 \times 3 = 2^1 \times (2^2 - 1)$$

$$28 = 4 \times 7 = 2^2 \times (2^3 - 1)$$

$$496 = 16 \times 31 = 2^4 \times (2^5 - 1)$$

Il démontra, ainsi que si $M = 2^p - 1$ est premier, alors $M(M + 1)/2 = 2^{p-1}(2^p - 1)$ est parfait.

- Au XVII e siècle, Le mathématicien Marin Mersenne étudia les nombres premiers de la forme pour tout entier k, M = $2^k - 1$.
- Un siècle après, Euler que tout nombre parfait pair est de la forme proposée par Euclide.
- Tout nombre parfait pair se termine par un 6 ou un 8.
- Tous les nombres parfaits pairs, excepté le premier, sont la somme des $2^{(n-1)/2}$ premiers cubes impairs :

  $28 = 1^3 + 3^3$

  $496 = 1^3 + 3^3 + 5^3 + 7^3$
- Leur nombre est aujourd'hui est de 49 nombres parfaits pairs, le dernier fut découvert en janvier 2016.

III. Les nombres parfaits impairs

« Il n'existe pas de nombre parfait impair » cette hypothèse est considérée comme vraisemblable mais pas encore prouvée. A ce jour, il a été établi que si un tel nombre existe, il possèderait au moins 300 chiffres.

III.1 Quelques résultats utiles :

- Euler a montré que si un tel nombre N existait alors il s'écrirait comme suit :

$$\begin{cases} N = \prod_{i=0}^{t} p_i^{a_i} \\ a_i \equiv 0\ [2] \\ p_0 \equiv a_0 \equiv 1\ [4] \end{cases} \quad (1)$$

Avec $p_0 \dots p_t$ des premiers impairs, où, $a_0 \equiv 1\ [4]$ et pour i = 1 … t.

Les $p_i^{a_i}$ sont appelés les composantes de N et le premier p_0 est appelé le premier spécial de N.

IV. Problématiques :

A ce jour, aucun nombre parfait impair n'a été trouvé, mais on n'a pas pu montrer non plus qu'il n'en existe aucun. La recherche des nombre parfaits impairs a été faite en montrant qu'il n'en existe aucun inférieur à une certaine borne donnée K.

En 1957, *Kanold* l'a fait pour K = 10^{20}, et des améliorations ont été donnés par *Tuckerman* (K = 10^{36}) et *Hagis* (K = 10^{50}), aujourd'hui cette borne avoisine les 10^{1500},

Mais dans ce projet, la borne choisie est 10^{160}, en adoptant une approche algorithmique pour démontrer le théorème suivant :

« Il n'y a pas de nombre parfait impair inférieur à 10^{160} *»* *(*)*

La borne choisie 10^{160} est reliée à l'état de l'art dans les méthodes de factorisations, dans lesquelles les nombres de 80 chiffres sont accessibles, mais pour les larges factorisations demandées dans la suite, nous utiliserons des outils adéquats.

II Présentation et compréhension de l'article :

I. Présentation :

Notre projet se base sur l'étude de la partie : ***New Lower Bound for Odd Perfect Numbers*** de l'article « *MATHEMATICS OF COMPUTATION VOLUME 53, NUMBER 187, PAGES 431-437* » qui a été écrit en juillet 1989 par : *Richard P.Brent* (Mathématicien et informaticien australien né en 1946) et *Graeme L. Cohen*.

Dans cet article, une méthode permettant de générer – sous la forme d'un arbre de contradictions – une preuve qu'un nombre parfait impair ne peut pas être inférieur à 10^{160} .

Pour l'écriture de cet article, plusieurs propriétés et résultats ont été utilisés ; Nous allons détailler dans ce qui suit ceux qui nous ont été utiles pour la compréhension de la méthode utilisée pour la preuve du théorème cité plus haut.

II. Compréhension :

- Les diviseurs de N :

D'après la définition d'un nombre parfait N, on a $\sigma(N) = 2N$, et puisque la fonction σ est multiplicative (i.e. $\sigma(p.q) = \sigma(p).\sigma(q)$) et rappelons que $\sigma(p^a) = 1 + p + \cdots + p^a$ alors l'ensemble des diviseurs de 2N et l'ensemble des diviseurs de $\sigma(N)$ sont égaux

$$\{q : q \text{ premier}, q \geq 2, q \mid \sigma({p_i}^{ai}) \text{ pour } i=0,1,\ldots,t\} = \{p_0,\ldots,p_t\} \qquad (2)$$

Associé à ce résultat, une méthode efficace de factorisation, alors de chaque composant p^a de N, on peut générer d'autres facteurs premiers de N, à partir des facteurs premiers impairs de $\sigma(p^a)$.

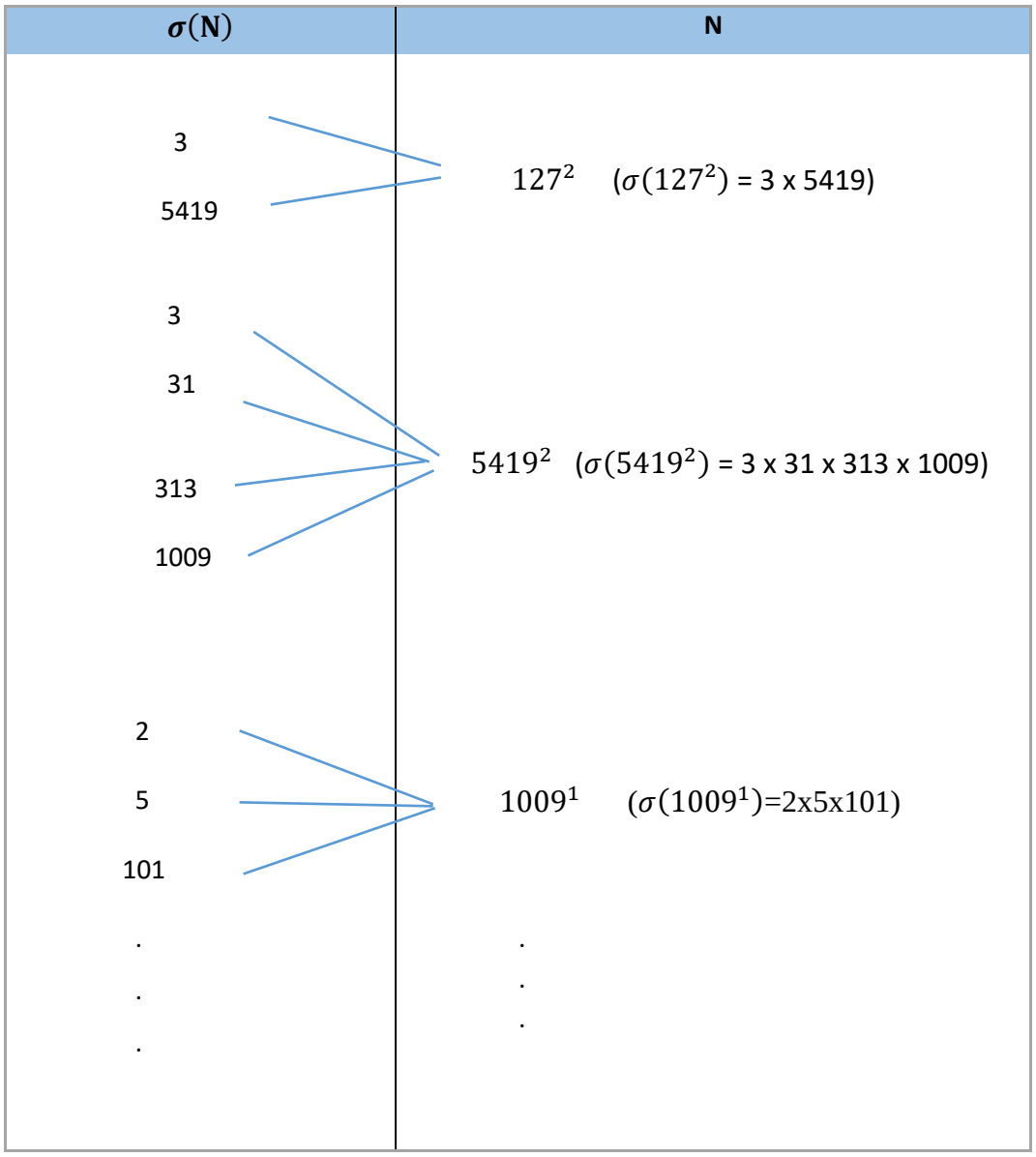
Supposons que 127^2 est l'une des composantes de N, et puisque $\sigma(127^2) = 1 + 127 + 127^2 = 16257$ et la décomposition de 16257 en facteur premier est 1625 = 3 x 5419, donc 3 et 5419 sont les facteurs premiers impairs de $\sigma(127^2)$, mais ils sont aussi les facteurs premiers de N à exposant ajusté (d'après (2)), de façon à ce que l'écriture d'Euler (1) soit vérifiée.

Ainsi un certain facteur premier q de N généré comme précédemment, on applique un exposant b et on factorise $\sigma(q^b)$ pour pouvoir continuer à générer d'autres facteurs premiers de N. On continue le processus de cette manière jusqu'à ce qu'on retombe sur une contradiction ou bien jusqu'à ce qu'un nombre parfait impair est trouvé.

En général, le facteur premier q de N auquel on réapplique le processus est choisi de façon à ce qu'il soit le plus grand des facteurs obtenus à chaque étape, afin de tomber plus rapidement sur une contradiction. Rien n'empêche de choisir d'autres facteurs, cependant avec des petits facteurs on va étaler la procédure.

De même, en utilisant la définition d'un nombre parfait N, on remarque que 2 divise $\sigma(N)$. Et puisque N est impair, alors 2 divise exactement $\sigma(N)$. En effet, le diviseur : 2 est ramené par le premier spécial p_0 puisque $\sigma(p_0)$ est pair ($\sigma(p_0) = p_0$+1 et p_0 est impair).

Pour plus d'explication du processus, nous avons développé dans le tableau suivant un exemple en supposant au début que 127^2 est un diviseur de N :



- Les exposants :

Le choix des exposants des diviseurs est basé sur la condition de l'écriture d'*Euler (1)* , ainsi que sur le résultat suivant :

Propriété :

Pour tout p premier :

$$a + 1 \mid b + 1 \Rightarrow \sigma(p^a) \mid \sigma(p^b) \qquad (3)$$

Preuve :

Supposons que $a + 1 \mid b + 1$, alors $\exists\ k \in Z$ tel que $b + 1 = (a + 1).k$

Donc : $\sigma(p^{b+1}) = \sigma\left(p^{(a+1)k}\right) = \sigma(p^{a+1})^k = \sigma(p^{a+1}).\sigma(p^{a+1})^{k-1}$

Ainsi : $\sigma(p^{a+1}) | \sigma(p^{b+1})$

Et puisque : $\sigma(p^{b+1}) = \sigma(p^b)\sigma(\mathrm{p})$, et de même pour $\sigma(p^{a+1})$

Alors : $\sigma(p^a)\sigma(\mathrm{p}) | \sigma(p^b)\sigma(\mathrm{p})$, d'où : $\sigma(p^a) | \sigma(p^b)$

Corollaire 1 :

L'exposant a doit être choisie de telle sorte que $a + 1$ soit premier. (4)

Preuve :

Pendant le processus, si on applique un exposant a à un premier p, on suppose que p^a est un diviseur exact de N, arrivé un certain temps ou une contradiction apparait, on peut en conclure que notre supposition est fausse. Dans certain cas (cela dépend du type de contradiction) on peut dire que p^a n'est pas un diviseur exact, mais rien n'empêche qu'il soit un diviseur. Cependant, ce processus sert à traiter tous les cas possibles, donc on applique un exposant plus grand. En exigeant que $a + 1$ soit premier, on évite le fait de retomber sur un cas déjà traité et donc on évite une redondance de calcul.

Exemple :

Dans le cas où $a = 2\ et\ b = 8$, on a $2 + 1 \mid 8 + 1$ et donc $\sigma(p^2) \mid \sigma(p^8)$.

Ainsi, si p≢1[4], on applique d'abord l'exposant 2 à p qui pourra ensuite être appliquer à la fonction adéquate.

Par contre, si à un certain moment on tombe sur une contradiction qui permet de conclure que p^2 n'est pas un diviseur exact, on passe alors au traitement du cas où on applique 4 comme exposant avant de réappliquer le processus (car d'après la condition d'Euler, l'exposant doit être pair) et puis on passe à 6 et puis 8. En factorisant $\sigma(p^8)$ on retombe certainement sur $\sigma(p^2)$ comme diviseur qui est un cas qui est déjà traité.

Corollaire 2 :

De plus, on peut supposer que $a_0 = 1$ (a_0 étant l'exposant du premier spécial).

Preuve :

Car sinon, et puisque $a_0 \equiv 1[4]$, alors : $a_0 + 1 \equiv 2[4]$, or on a exigé le choix de l'exposant de telle sorte que cette exposant plus une unité, soit premier. Le seul cas où a_0 vérifie $a_0 + 1$ premier et $a_0 + 1 \equiv 2[4]$ est lorsque $a_0 = 1$.

Remarque :

Soit p premier et a un entier strictement positif vérifiant $a + 1$ premier; Les diviseurs premiers possibles de $\sigma(p^a)$ sont exactement $a + 1$ si et seulement si $p \equiv 1\ [a + 1]$, et les premiers $q \equiv 1\ [a + 1]$.

- Le théorème :

Propriété :

Pour montrer qu'il n'existe pas de nombre parfait impair N inferieur à 10^{160},il suffit de montrer qu'aucun des nombres premiers cités ci-dessous n'est diviseur de ce nombre N.

$$127, 19, 7 , 11 , 31 , 13 , 3 , 5 \qquad (5)$$

L'ordre est ainsi donné afin d'optimiser les factorisations.

Preuve :

Si aucun des premiers suggérés (5) n'est un facteur de N, alors N a certainement au moins 101 facteurs premiers distincts.

Car s'il y'en avait moins, alors :

$$\frac{\sigma(N)}{N} = \prod_{i=0}^{t} \frac{\sigma(p_i^{a_i})}{p_i^{a_i}} = \prod_{i=0}^{t} \left(\frac{\frac{1-p_i^{a_i+1}}{1-p_i}}{p_i^{a_i}}\right) = \prod_{i=0}^{t} \frac{-p_i + p_i^{-a_i}}{1-p_i} = \prod_{i=0}^{t} \left(1 + \frac{-1+p_i^{1/a_i}}{1-p_i}\right)$$

$$= \prod_{i=0}^{t} \left(1 + \frac{p_i^{-a_i}-1}{1-p_i}\right) \leq \prod_{i=0}^{t} \left(1 + \frac{-1}{1-p_i}\right) = \prod_{i=0}^{t} \left(1 + \frac{1}{p_i-1}\right) = \prod_{i=0}^{t} \frac{p_i}{p_i-1}.$$

En prenant les cents premiers nombres premiers, tel que P est l'ensemble des premiers satisfaisant $37 \leq p \leq 599$, avec $p \neq 127$:

$$\frac{\sigma(N)}{N} < \prod_{i=0}^{t} \frac{p_i}{p_i-1} \leq \frac{17}{16}\frac{23}{22}\frac{29}{28}\prod_{P} \frac{p}{p-1} < 2 .$$

L'abondance n'atteignant pas la valeur de deux, prendre les cent nombres premiers cités précédemment ne suffit pas, il faut donc, au moins, 101 termes.

On a ainsi :

$$N \geq \left(17^2.23^2.29^2.\prod_{P} p^2\right).601^1 > 10^{473}$$

On cherche une minoration de la valeur de N, alors on choisit l'exposant impair « 1 » au plus grand nombre premier, qui est 601 et le reste des nombres premiers auront l'exposant pair « 2 ».

Donc l'élimination des huit nombres premiers (5) mentionnés comme facteurs de N est la preuve du théorème (*).

- Les contraintes :

L'ensemble des contradictions résultants d'un choix de premiers et d'exposants pendant chaque étape du processus constitue une preuve du théorème. Chaque contradiction est une contrainte qui arrêtera le processus appliqué à un certain premier p. Et selon son type, la contrainte impliquera soit le changement de l'exposant du premier p, soit le changement de l'exposent du diviseur premier précédant p.

Dans cette partie, on explicitera l'ensemble de ces contraintes dans un ordre optimal au processus :

La contrainte B_0: *[$\sigma(p^a)$ a un diviseur qui a déjà été considéré]*

Si au cours du processus, en factorisant le résultat de la somme des diviseurs d'un certain diviseur premier p avec un certain exposant a, on retombe sur l'un des diviseurs déjà considéré dans l'une des étapes précédentes du processus comme un diviseur exacte, alors dans ce cas on aboutit à une contradiction. Car le fait de considérer ce premier p à une puissance fixée a (i.e. p^a) comme étant un diviseur exact de N signifie que p^a|N et que $p^{a+1} \nmid N$. Ainsi, retrouver ce diviseur p contredit le fait que p^a est un diviseur exact. On appelle cette Contradiction la contrainte B_0.

En retombant sur cette contrainte, on en déduit que p^a n'est pas un diviseur exact, mais rien ne montre que p n'est pas un diviseur. Ainsi, on arrête le processus et on passe à l'exposant suivant de p (en prenant en considération que l'exposant suivant doit vérifier la condition (4)).

La contrainte D : *[$\sigma(p^a)$ a un diviseur qui a déjà été éliminé]*

Cette contrainte est testée après la factorisation de $\sigma(p^a)$. C'est une contradiction de l'hypothèse : « p^aest un diviseur exact de N ». Elle indique, si elle est vérifiée, que l'un des diviseurs de $\sigma(p^a)$ a précédemment été éliminé par une des contradictions qu'on introduira par la suite. Ainsi le reconsidérer comme diviseur impliquera une redondance de calcul avant de retomber sur la même contrainte qui l'a éliminé en premier lieu.

La contrainte B_2 : *[$p^a > 10^{80}$]*

Pour cette contrainte, on utilise le résultat suivant :

Propriété :

Si p^a est un diviseur de N, alors $p^a < 10^{80}$.

Preuve :

Supposant par absurde que : $p^a > 10^{80}$

En effet, on a : $N \geq p^a \sigma(p^a)$,

Or : $\sigma(p^a) > p^a$(puisque : $\sigma(p^a) = 1 + p + \cdots + p^a$)

Et donc : N $> p^{2a} > 10^{160}$

On aboutit à une contradiction puisqu'au départ nous avons supposé que N est un nombre premier impair inférieur à $(10)^{160}$.

Cette propriété permet de limiter le nombre des exposants à considérer et de garantir que le changement des exposants impliqués pendant le processus par la vérification de certaines contraintes s'arrêtera à un moment donné.

Ainsi, au cours du processus, on continue à développer la chaine de factorisation et en retombant sur la contrainte $\underline{B_2}$ i.e. un composant p^a qui dépasse 10^{80}, on arrête le processus et on revient au composant précédant auquel on effectue un changement d'exposant (selon la condition (4)). Puisque cette contrainte ne montre pas seulement que p^a n'est pas un diviseur exact, mais que p^a n'est carrément pas un diviseur de N (ça ne sert donc à rien de chercher les exposants suivants). C'est une contradiction de l'hypothèse : « p^aest un diviseur de N ».

La contrainte B_3 :

[Le produit des composantes de N (sans les facteurs de $\sigma(p^a)$ en cours) $> 10^{160}$*]*

Cette contrainte est vérifiée si le produit de tous les diviseurs premiers de N générés pendant le processus (sans prendre en compte les facteurs de $\sigma(p^a)$ en cours de calcul), avec leurs exposants ajustés en l'augmentant de façon à ce qu'il soit conforme avec la forme d'Euler et la condition (4), dépasse la borne 10^{160} .

Si cette contrainte est vérifiée pour p^a, c'est que non seulement p^a n'est pas un diviseur exact de N mais que ce n'est pas un diviseur de N. Du fait que la contrainte est vérifiée au moins pour p^a, elle sera vérifiée aussi en appliquant les exposants suivants. Ainsi, c'est une contradiction de l'hypothèse : « p^a est un diviseur de N ». On arrête donc le processus et on revient au composant précédant auquel on effectue un changement d'exposant (selon la condition (4)).

La contrainte B_1 : *[Le produit des composants de N $>$ 10^{160}]*

Cette contrainte est presque similaire à la contrainte B_3 . Elle est vérifiée si le produit de tous les diviseurs premiers de N générés pendant le processus, avec leurs exposants ajustés en l'augmentant de façon à ce qu'il soit conforme avec la forme d'Euler et la condition (4), dépasse la borne 10^{160} .

Contrairement à B_3, elle est donc testée en incluant les facteurs de $\sigma(p^a)$ de p^a en cours à exposants ajustés. Ainsi, si cette contrainte est vérifiée pour p^a, c'est que p^a n'est pas un diviseur exact de N. Cependant, rien ne garantit que ce n'est pas un diviseur de N. Il s'agit donc d'une contradiction de l'hypothèse : « p^a est un diviseur exact de N ». On arrête donc le processus et on effectue un changement d'exposant (selon la condition (4)). Cette contrainte est alors plus faible que la contrainte B_3.

La contrainte S : *[* $\prod \frac{\sigma(p^{ai})}{p^{ai}} > 2$*]*

Certaines chaines constituées au cours du processus peuvent être terminée si

$\prod \frac{\sigma(p^{ai})}{p^{ai}} > 2$. En effet, plusieurs premiers petits qui ont été généré, et dont le produit à exposants ajustés en convenance avec la forme d'Euler et la condition (5) est un nombre m qui satisfait :

$$S = \frac{\sigma(\mathrm{m})}{m} > 2$$

Tout diviseur l de N doit satisfaire $\sigma(\mathrm{l})/l \leq 2,$ de telle sorte à ce que ce « S-test » donne une autre contrainte d'arrêt. C'est une contradiction de l'hypothèse : « p^aest un diviseur exact de N », Car elle vérifie uniquement que p^a n'est pas un diviseur exact. On arrête donc le processus et on effectue un changement de composant (selon la condition (4)).

III Le programme en C :

I. Quelques outils :

L'un des premiers problèmes auquel on a fait fasse lors de l'implémentation de ce qu'on a expliqué précédemment, est la manipulation et factorisation de grands nombres, mais bien heureusement pour nous, il existe des outils très efficaces pour répondre aux besoins de l'implémentation.

- Gestion de grands nombres :

La bibliothèque GMP (GNU Multiple Precision arithmetic library) permet la gestion de grands entiers, elle fournit de nombreuses fonctions de calcul sur différents types :

- Grands entiers Z
- Grands rationnels Q
- Grands flottants R

Elle a donc son propre type de variables « mpz_class » et offre tout un panel de fonctions sur la somme, la multiplication, la division, l'exponentiation, la comparaison …etc.

- Factorisation de grands nombres :

La factorisation d'entier est un problème fondamental en algorithmique arithmétique, et dans ce projet, nous avons fait face à de grands nombres à devoir factoriser sans connaitre a priori la taille de leur diviseur. Tout en ne négligeant pas la méthode naïve, car quel que soit la taille d'un diviseur, il reste très probable qu'il ait un petit diviseur strict ; Cette méthode consiste en la division de ce grand nombre par les 10000 premiers nombres premiers afin de pouvoir utiliser sur le grand diviseur restant, selon sa taille, un algorithme plus performant tel que : GNFS (General Number Field Siev) ou ECM (Elliptic Curve Method).

D'une part, il y a GNFS, l'algorithme de factorisation par crible sur corps de nombres, il a été introduit par *Pollard* en 1988, c'est l'une des méthodes les plus rapides, mais

dans ce projet, on ne s'y intéressera pas, en raison de ses caractéristiques qui ne conviennent pas au besoin du projet.

D'autre part, il y a l'algorithme ECM (Elliptic Curve Method), un algorithme de factorisation proposé par *H. Lenstra* en 1987, il utilise les propriétés des courbes elliptiques sur les corps finis pour factoriser des entiers.

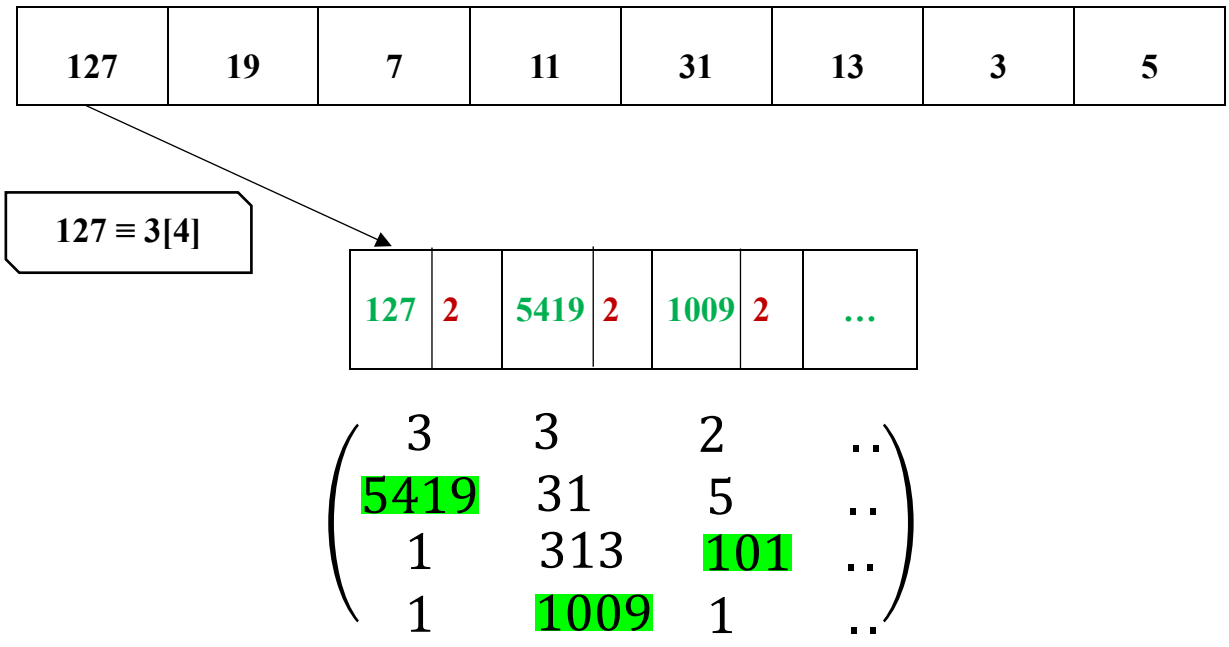
Il représente le meilleur algorithme d'extraction de diviseurs, de taille maximale : 64 bits, puisque son temps de calcul dépend de la taille du facteur à trouver et non pas de la taille de l'entrée, contrairement à l'algorithme précédent.

Pour la suite du projet, concernant les parties de factorisations expliquées dans la deuxième partie, nous utiliserons la bibliothèque ecm qui nous permet d'appliquer cet algorithme.

II. Le programme :

Comme expliqué précédemment, la preuve du théorème (*) est sous la forme d'un arbre de contradictions.

Nous réécrivons l'exemple précèdent comme suit :



A partir de là, nous avons décidé de représenter les 8 nombres premiers à tester sous forme de tableau, tout comme les nombres premiers sur lesquels on fera les hypothèses, on représente chaque nombre premier et son exposant dans une cellule, ce qu'on va nommer les éléments à exposant fixe. À partir de chacun de ces éléments à exposant fixe, on génèrera des diviseurs premiers en appliquant la fonction sigma (somme des diviseurs) et nous testerons les différentes contraintes exposées précédemment, jusqu'à ce que l'une d'entre elles soit validée.

Présentation de quelques fonctions utilisées dans le programme :

Les éléments à exposant fixe sont représentés par une structure qui prend en paramètre : un nombre premier de type mpz_t et un exposant de type unsigned long int.

```
struct premier_exposant
{
    mpz_t z_premier;

    unsigned long int exposant;
};
```

A partir du tableau des huit nombres premiers, on teste l'écriture d'Euler (1) avec la fonction suivante :

```c
int test_sur_composante(mpz_t z_premier)
{
        mpz_t z_r,z_d,z_c;
        mpz_init (z_r);
        int resultat;
        mpz_init_set_ui (z_d,4);
        mpz_init_set_ui (z_c,1);
        mpz_mod (z_r, z_premier, z_d);
        resultat = mpz_cmp (z_r, z_c);
        if (resultat == 0)
        {
                printf("\nIl faut prendre l'exposant '1', ce nombre peut être le premier spécial\n");
                return 1;
        }
        else
        {
                printf("\nIl faut prendre un exposant pair\n");
                return 0;
        }

}
```

Cette fonction *test_sur_composante* renvoie 1 si le nombre est le premier spécial sinon il renvoie 0 ; Dans ce dernier cas, nous avons une fonction qui génère les exposants pairs à appliquer à ce nombre premier :

```c
void tableau_exposant (unsigned long int exposant[])
{
        mpz_t z_e,z_p;
        mpz_init_set_ui(z_e,2);
        mpz_init(z_p);
        int entier_p,cmpt_expo_list;
        for (cmpt_expo_list = 0;cmpt_expo_list < 40; cmpt_expo_list++)
        {
         mpz_nextprime (z_p,z_e);
         entier_p = mpz_get_ui(z_p);
         exposant[cmpt_expo_list] = entier_p-1;
         mpz_set(z_e,z_p);
         //printf("\nl'exposant à la case %i est: %li \n",cmpt_expo_list,exposant[cmpt_expo_list]);
        }
}
```

Cette fonction *tableau_exposant* remplit un tableau avec les exposants pairs appropriés, en utilisant la fonction de la bibliothèque gmp *mpz_next_prime*, qui prend en paramètre

un nombre de type mpz_t et qui renvoie le nombre premier qui le suit, on vérifie si ce nombre premier est égal à l'exposant moins une unité.

La taille du tableau est définie par la contrainte B2, en prenant le pire des cas, il faut que

$3^x \leq 10^{80} <=> e^{\ln(3)^x} \leq e^{\ln(10)^{80}} <=> e^{xln(3)} \leq e^{80\ln(10)} <=> xln(3) \leq 80ln(10)$

$<=> x \leq \frac{80\ln(10)}{\ln(3)} \approx 167{,}67.$

L'exposant x doit donc approcher les 167,67 de telle sorte à ce que x + 1 = p ou p est un nombre premier. Dans ce projet, le plus grand exposant pair est donc : 178.

Après que l'exposant soit choisi, on vérifie la contrainte B2, en calculant la valeur du nombre premier élevé à la puissance choisie, en utilisant une fonction proposée par la bibliothèque gmp : *mpz_pow* et comparant ce résultat à la racine de la borne.

```
int contrainte_b2(mpz_t z_new,unsigned long int expo_new)
{
        mpz_t z_b2;
        mpz_init_set(z_b2,z_new);
        mpz_t z_racine_borne;
        mpz_init_set_ui(z_racine_borne,10);
        int cmp_b2;
                mpz_pow_ui(z_racine_borne, z_racine_borne, 80);
                mpz_pow_ui(z_b2, z_new, expo_new);
                cmp_b2=mpz_cmp(z_b2,z_racine_borne);
if ((cmp_b2==-1)||(cmp_b2==0)) { return 0; }
else { return 1; }
}
```

Si une contradiction apparait, alors on efface ce nombre et la colonne des diviseurs lui correspondant dans la matrice, et on passe, l'élément à l'exposant fixe le précédent, à l'exposant suivant du *tableau_exposant.*

Sinon le calcul continue, à partir du nombre premier et de l'exposant, on calcule la somme de ces diviseurs grâce à la fonction suivante :

```c
void mpz_sigma(mpz_t z_somme,mpz_t z_premier_N, int exposant)
{
    int i;
        i = 0;
        mpz_t resultat_expo;
        mpz_init_set_ui(resultat_expo,0);
        mpz_init_set_ui(z_somme,0);

    for (i = 0 ; i <= exposant ; i++)
    {
     mpz_pow_ui(resultat_expo, z_premier_N, i);
     mpz_add(z_somme,resultat_expo,z_somme);
   }

return;
}
```

Rappel :

Soit p un nombre premier et a un entier naturel positif, $\sigma(p^a) = 1 + \mathrm{p} + \cdots + p^a$.

Cette fonction prend en entrée un nombre premier de type mpz_t et un exposant de type int, elle utilise la fonction d'exponentiation de la bibliothèque gmp et renvoie la somme des diviseurs de p^a.

L'étape de la factorisation se fait en deux temps :

Dans un premier temps, Nous vérifions si le résultat de la fonction *mpz_sigma* n'est pas premier, en utilisant une fonction de la bibliothèque gmp *: gmp probab_prime*, cette fonction renvoie : 2, 1 ou 0 dans le cas où le nombre est certainement premier, probablement premier ou composé. S'il est premier, l'élément est écrit dans la matrice, sinon on applique le résultat de la fonction *mpz_sigma* à une fonction *factoriser*, qui vérifie si ce résultat n'a pas de facteur premier avoisinant les 10^5, nous avons, au préalable, un fichier contenant les 10 000 premiers nombres premiers mis dans un tableau.

```
void remplissage_tableau_10000_premier(mpz_t tableau_10000_premier[])
{
        int i;
        unsigned long int variable_temporaire;

        FILE* fichier = NULL;
        fichier = fopen("premier.txt", "r");
        if (fichier != NULL)
        {
                for(i = 0; i < 10000 ;i++)
                {
                mpz_init(tableau_10000_premier[i]);
                fscanf(fichier, "%ld", &variable_temporaire);
                mpz_set_ui(tableau_10000_premier[i],variable_temporaire);

                }
        }
        fclose(fichier);
}
```

Et dans un deuxième temps, nous utilisons une fonction de la bibliothèque ecm : *ecm_factor*, qui, comme on l'a précisé précédemment, utilise les courbes elliptiques, cette fonction prend en paramètre le nombre à factoriser, la valeur d'une variable appelé B_1 et l'option NULL pour l'initialiser d'un pointeur vers une structure pour une certaine option de la factorisation et renvoie un entier :

- 0 s'il ne trouve pas de diviseur
- 1 si la fonction a trouvé une courbe gagnante à l'étape 1
- 2 si elle a trouvé une courge gagnante à l'étape 2

La valeur de B_1 est choisie selon la taille de l'entrée, plus cette valeur est grande, plus la probabilité de de trouver un grand facteur est forte, mais en augmentant B_1, le temps de calcul augmente aussi.

- Etape 1 : B_1 étant un paramètre obligatoire, il peut être donné en format entier ou à virgule flottante. Sa plus grande valeur avoisine les 10^{15},tous les nombres premiers compris entre 2 et la valeur de B_1sont traité à l'étape 1.
- Etape 2 : B_2 est un paramètre optionnel, s'il est supprimé, alors une valeur par défaut est calculée à partir de B_1 ; Comme B_1, sa valeur peut être un entier ou en virgule flottante. Sa plus grande valeur avoisine 10^{23}, tous les nombres premiers compris entre B_1 et B_2 sont traités à l'étape 2.

Exemple :

Soient p_1 et p_2 deux nombres premiers, respectivement, de 18 et 25 chiffres $p_1 = 866819693638854967$ et $p_2 = 1159508627333214596192701$, on calcule leur produit n de 43 chiffres et on le met en entré de *ecm_factor* avec un $B_1 = 1\,000$, le facteur p_1 est trouvé à la deuxième étape, au bout de 4 ms et de 756 courbes elliptiques.

Alors qu'avec un $B_1 = 10\,000$, ce même facteur est trouvé à l'étape 2, au bout de 20 ms et de 80 courbes elliptiques

Et avec un $B_1 = 100\,000$, $p_1$est trouvé à l'étape 2 au bout de 208 ms et à la sixième courbe.

Dans le programme, la valeur initiale de B_1 vaut 75 000 et est multipliée par deux à chaque facteur trouvé par *ecm_factor* .

Tout au long de ces factorisations, à chaque résultat positif, on remplira la matrice de ces diviseurs, tout en vérifiant leur occurrence ; Selon la première méthode de factorisation, on sait que les diviseurs, s'ils existent, apparaitront dans un ordre croissant, ce qui nous permet de vérifier, si à chaque diviseur trouvé, il n'est pas égal au diviseur le précédant.

Concernant la factorisation avec *ecm_factor*, nous supposons que le diviseur trouvé, étant un « grand diviseur » il est peu probable qu'il apparaisse plus d'une fois.

La matrice contenant les diviseurs est de taille 45 x 45 ; Le nombre de diviseurs p, sans prendre en compte leur multiplicité est limité puisque la borne choisie est de l'ordre de 10^{160}, en posant le calcul suivant :

$$(\prod p)^2 . 17 = 8.78 * 10^{160} \qquad pour\ tout\ nombre\ premier\ 3 \leq p \leq 199.$$

A ce stade-là, nous avons une matrice contenant des diviseurs qu'on transfère dans un tableau trié de taille 2025.

Pour pourvoir appliquer les contraintes restantes, nous devons ajuster les exposants de ces diviseurs, si par exemple un facteur est trouvé trois fois, alors son exposant doit être « 4 » afin qu'il vérifie l'écriture d'un nombre parfait selon Euler.

Comme nous l'avons précisé dans la partie « explication des contraintes », nous les testerons dans un ordre précis, afin d'éviter un maximum de calcul en retombant sur des contradictions dans l'arbre.

Concernant les contraintes suivantes, pour éviter la perte d'informations dans la matrice, comme dit précédemment, nous les copierons à chaque étape dans un tableau trié, et nous ajusterons les exposants selon les règles d'écriture d'Euler.

Tant qu'un nombre vérifiant les propriétés d'un premier spécial n'est pas trouvé, il n'y aura, dans N, que des éléments à exposant pair ; et si plusieurs nombres vérifient la propriété alors, dans un premier temps, dans la matrice, tous ces éléments auront un exposant impair (qui vaut 1), mais dès qu'une hypothèse est posée sur le plus grand des nombres « premier spécial » alors il n'en existera qu'un seul dans N, donc tous les autres nombres, auront un exposant pair ajusté dans le tableau.

Le premier test sur la matrice est la contrainte B3(uniquement si la contrainte B2 n'est pas vérifiée), nous remplirons le tableau avec les éléments de la matrice sauf la dernière colonne, après le tri et l'ajustement des exposants, il nous suffira de faire le produit de tous ses éléments et des éléments du tableau contenant les éléments à exposant fixe (les éléments supposés être des diviseurs exacts) ; Si le résultat du produit est inférieur à 10^{160}, alors le processus continue, sinon une contradiction apparait, nous effaçons la dernière colonne de la matrice ainsi que le diviseur supposé exact, et une nouvelle hypothèse est émise sur le diviseur exact précèdent le diviseur exact effacé.

Remarque :

A chaque colonne, il y a un diviseur exact, sur lequel on posera une hypothèse, donc ce diviseur apparaitra dans la matrice et dans le tableau des éléments à exposant fixe ; A la création tableau ce diviseur est pris en compte qu'une seule fois.

```
int contrainte_b3(mpz_t z_b3)
{
        int comp;
        mpz_t z_m;

        mpz_init(z_m);
        mpz_set_ui(z_m,10);

        mpz_pow_ui(z_m,z_m,160);

        comp = mpz_cmp(z_m,z_b3);
return comp;
}
```

Si nous ne rencontrons pas de contradiction de type B3 , on teste la fonction B1, qui comme B3, remplit un tableau des éléments de la matrice, sauf que pour B1, on prend tous les éléments de la matrice en ajustant les exposants. La fonction calcule le produit de ses éléments et des éléments à exposant fixe, s'il est supérieur à 10^{160}, il y a contradiction avec notre hypothèse de départ ($N < 10^{160}$) alors la dernière colonne de la matrice est effacée et l'exposant de l'élément qui lui est associé, dans le tableau des diviseurs supposée exacts, est modifié (il passe à l'exposant suivant).

Sinon le processus continue en vérifiant une autre contrainte, la S. La contrainte S est appliquée quand la contrainte B1 n'est pas vérifiée, elle reprend la formule vue dans la partie théorique et vérifie si l'abondance du nombre construit vaut 2.

Dans le cas où l'abondance est inférieure à deux, il n'y a pas de contradiction, le processus suit son cours et dans le cas contraire, l'apparition de cette contradiction induit la suppression de la dernière colonne de la matrice et la modification de l'exposant de l'élément qui lui est associé.

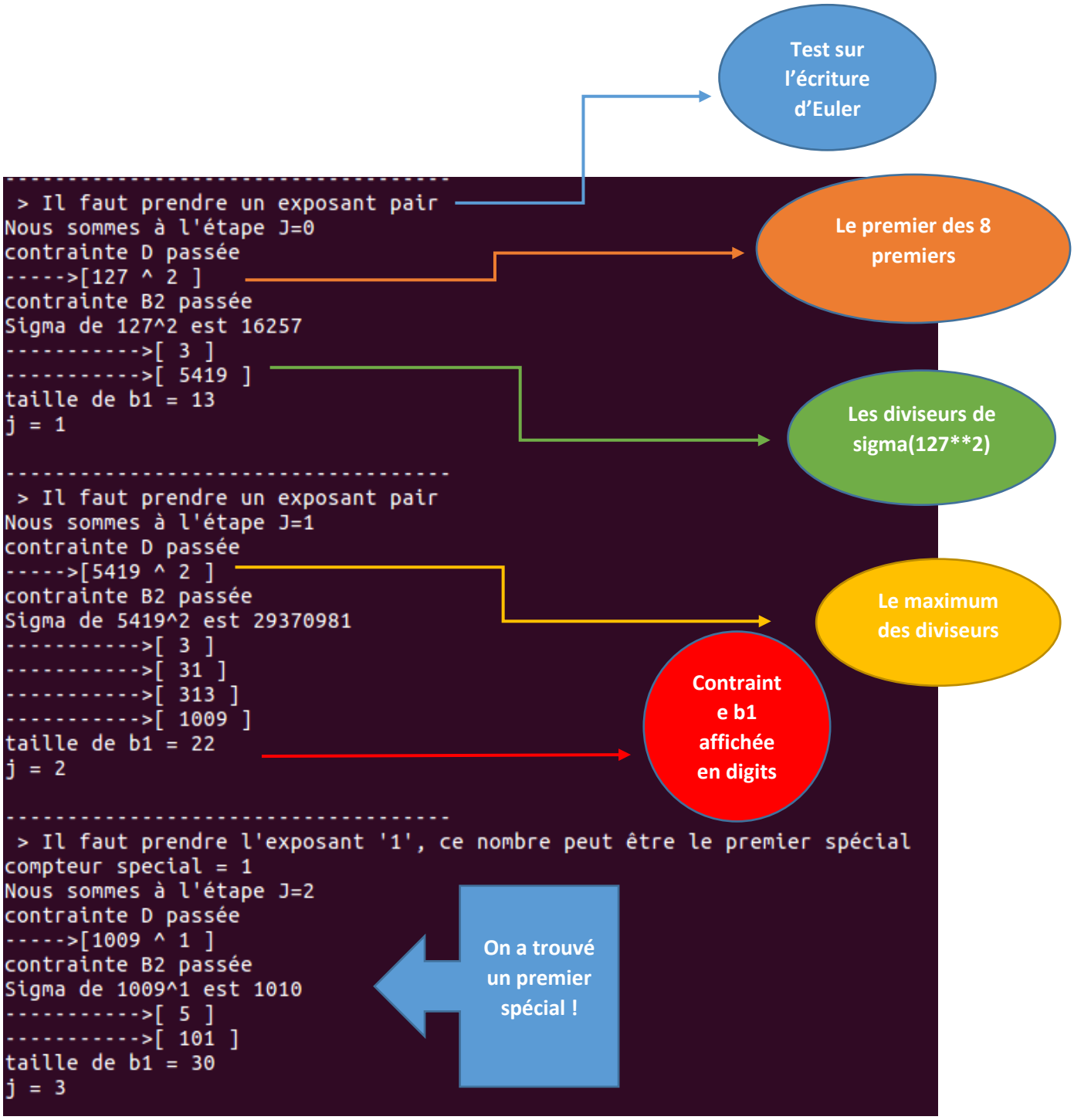
Nous appliquerons ces contraintes à chaque étape, jusqu'à remonter à la première hypothèse posée sur le nombre 127, le tout premier nombre premier des huit nombres premiers est éliminé, le processus recommencera en partant d'une autre hypothèse (à partir de 19) jusqu'à ce qu'elle ne soit plus valide (en rencontrant, en plus des contraintes précédentes, la contrainte D, qui elle est en forme de tableau contenant les éléments des huit nombres premiers déjà éliminés) ; Et ainsi de suite, le processus annulera les huit nombres premiers et c'est la preuve de l'inexistence d'un nombre parfait impair inferieure à 10^{160} .

L'exécution :

L'exécution du programme nécessite d'avoir les deux bibliothèques ecm et gmp ; son exécution est plutôt longue en vue des calculs qu'il fait, cela dépend surtout de la fonction ecm, par exemple l'élimination du nombre premier 127 peut atteindre deux heures d'exécution.

Les commandes d'exécution sont : ***gcc -Wall parfait.c main.c -lgmp -lecm -lm -o main*** et puis **.*main***

Voici un bout du programme :


Test sur l'écriture d'Euler
Le premier des 8 premiers
Les diviseurs de sigma(127**2)
Le maximum des diviseurs
Contraint e b1 affichée en digits
On a trouvé un premier spécial !

> Il faut prendre un exposant pair
Nous sommes à l'étape J=0
contrainte D passée
----->[127 ^ 2]
contrainte B2 passée
Sigma de 127^2 est 16257
----------->[3]
----------->[5419]
taille de b1 = 13
j = 1

> Il faut prendre un exposant pair
Nous sommes à l'étape J=1
contrainte D passée
----->[5419 ^ 2]
contrainte B2 passée
Sigma de 5419^2 est 29370981
----------->[3]
----------->[31]
----------->[313]
----------->[1009]
taille de b1 = 22
j = 2

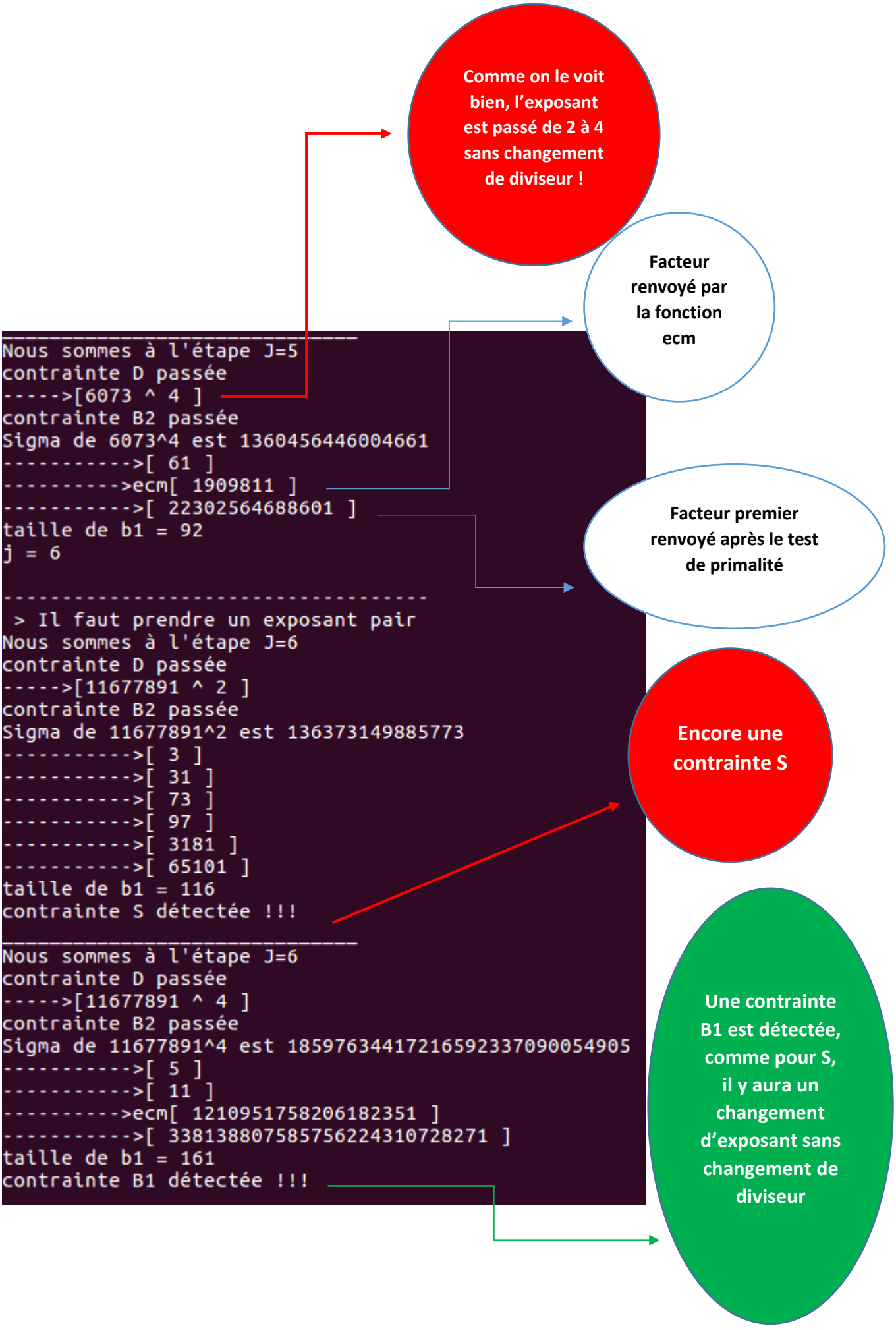
> Il faut prendre l'exposant '1', ce nombre peut être le premier spécial
compteur special = 1
Nous sommes à l'étape J=2
contrainte D passée
----->[1009 ^ 1]
contrainte B2 passée
Sigma de 1009^1 est 1010
----------->[5]
----------->[101]
taille de b1 = 30
j = 3

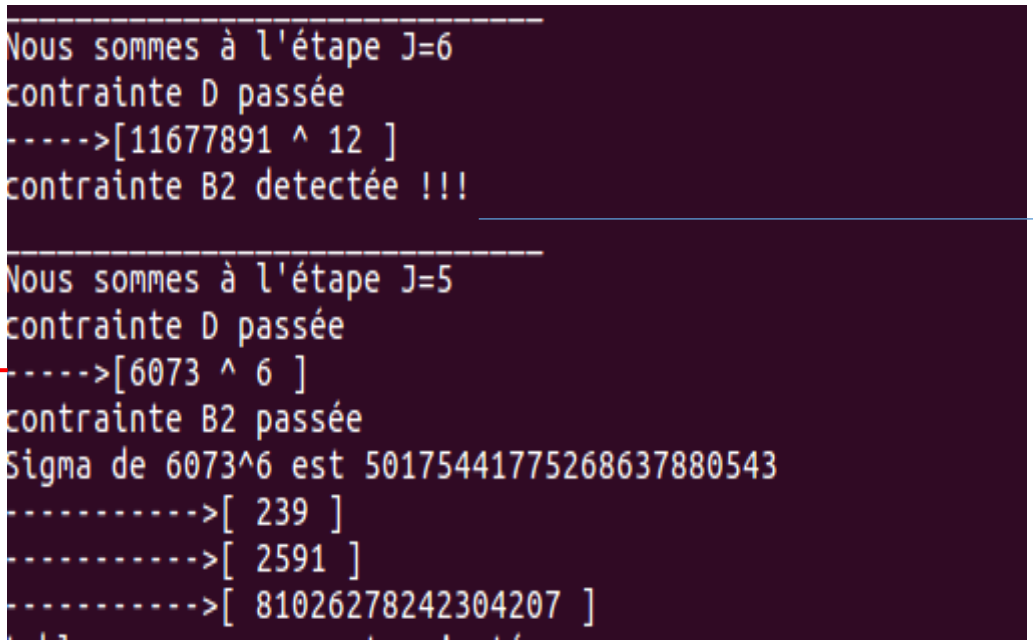
Le processus continue, en prenant le maximum des diviseurs et en appliquant les différentes contraintes

```
-----------------------------------
 > Il faut prendre un exposant pair
Nous sommes à l'étape J=3
contrainte D passée
----->[101 ^ 2 ]
contrainte B2 passée
Sigma de 101^2 est 10303
---------->[ 10303 ]
taille de b1 = 38
j = 4

-----------------------------------
 > Il faut prendre un exposant pair
Nous sommes à l'étape J=4
contrainte D passée
----->[10303 ^ 2 ]
contrainte B2 passée
Sigma de 10303^2 est 106162113
----------->[ 3 ]
----------->[ 5827 ]
----------->[ 6073 ]
taille de b1 = 54
j = 5

-----------------------------------
 > Il faut prendre un exposant pair
Nous sommes à l'étape J=5
contrainte D passée
----->[6073 ^ 2 ]
contrainte B2 passée
Sigma de 6073^2 est 36887403
----------->[ 3 ]
----------->[ 7 ]
----------->[ 139 ]
----------->[ 12637 ]
taille de b1 = 68
contrainte S détectée !!!
```

On detecteune contrainte de type S !

Changement d'exposant !!


Comme on le voit bien, l'exposant est passé de 2 à 4 sans changement de diviseur !
Facteur renvoyé par la fonction ecm
Facteur premier renvoyé après le test de primalité
Encore une contrainte S
Une contrainte B1 est détectée, comme pour S, il y aura un changement d'exposant sans changement de diviseur
Nous sommes à l'étape J=5
contrainte D passée
----->[6073 ^ 4]
contrainte B2 passée
Sigma de 6073^4 est 1360456446004661
----------->[61]
---------->ecm[1909811]
----------->[22302564688601]
taille de b1 = 92
j = 6
> Il faut prendre un exposant pair
Nous sommes à l'étape J=6
contrainte D passée
----->[11677891 ^ 2]
contrainte B2 passée
Sigma de 11677891^2 est 136373149885773
----------->[3]
----------->[31]
----------->[73]
----------->[97]
----------->[3181]
----------->[65101]
taille de b1 = 116
contrainte S détectée !!!
Nous sommes à l'étape J=6
contrainte D passée
----->[11677891 ^ 4]
contrainte B2 passée
Sigma de 11677891^4 est 18597634417216592337090054905
----------->[5]
----------->[11]
---------->ecm[1210951758206182351]
----------->[338138807585756224310728271]
taille de b1 = 161
contrainte B1 détectée !!!



On efface les diviseurs, et on emet une nouvelle hypothese sur le diviseur qui precede 11677891**12

Contrainte B ! On remonte d'une hypothèse.

On recommence avec 6073**6, on calcule la fonction sigma de 6073**6, on factorise, on récupère le maximum des diviseurs et on recommence

Conclusion

Ce projet autour des nombres parfaits, nous a permis de pouvoir appliquer certaines notions de programmation en langage C et surtout d'en découvrir, avec l'utilisation de nouvelles bibliothèques qui sont, la bibliothèque *gmp* et *ecm*. Leur utilité dans le projet, qui a été « la preuve de l'inexistence de nombres parfaits impairs inferieure à 10^{160} » a été primordiale.

La question de savoir si ces nombres existent n'a toujours pas de réponse, mais nous avons réussi, en reprenant l'article de *Richard P.Brent* et de *Graeme L. Cohen*, à l'aide d'outils informatiques et de preuve mathématiques à démontrer qu'il n'en existe pas d'inférieurs à 10^{160}.

De nos jours les ordinateurs ont offert une façon inédite d'approcher l'inconnu et avec cette évolution, la borne choisie a largement été dépassée, aujourd'hui nous avoisinons les 10^{2000} mais malgré le grand écart entre les bornes, ce projet nous été avantageux puisqu'il nous a permis, de voir en surface, ce qu'est l'exercice de recherche.

Au niveau de la gestion du projet en groupe, nous avons réussi à trouver une bonne répartition des tâches afin de réaliser ce projet dans les temps.

Ne pouvant pas commencer la programmation de cette preuve sans les explications théoriques, nous avons donc préféré avancer ensemble sur la partie lecture et compréhension de l'article, Concernant la partie de la programmation, Mazlani Kaoutar et Bedouhene Abderahmane se sont occupés de la partie algorithmique, Bedouhene abderhamane et Daoudi Sonia se sont occupé de la partie programmation et la rédaction et gestion d'informations ont été faites par Daoudi Sonia et Mazlani Kaoutar.

Bibliographie

Introduction

Paul Glendinning « Mathématiques minute » des éditions CONTRE-DIRES

http://www.univers-ti-nspire.fr/files/pdf/03-arith-parfaits-TNS21.pdf

http://www.maths-et-tiques.fr/index.php/histoire-des-maths/nombres/les-nombres-parfaits

http://fred.elie.free.fr/nombres_parfaits.pdf

I. Présentation générale des nombres parfaits :

http://serge.mehl.free.fr/anx/nb_parf.html

https://fr.wikipedia.org/wiki/Nombre_parfait

http://hamid-mira.blogspot.fr/2012/04/la-conjecture-des-nombres-parfaits.html

II. Présentation et compréhension de l'article :

http://www.unilim.fr/pages_perso/hieu.phan/PIR2016/Initiation_Recherche_M1maths_2016_2017_Clavier_OPN.pdf

http://maths-people.anu.edu.au/~brent/pub/pub100.html

III. Le programme en C :

https://fr.wikipedia.org/wiki/Crible_alg%C3%A9brique

http://maths-people.anu.edu.au/~brent/pub/pub100.html

http://defeo.lu/MA2-AlgoC/

https://fr.wikipedia.org/wiki/Factorisation_de_Lenstra_par_les_courbes_elliptiques

http://manpages.ubuntu.com/manpages/precise/man1/ecm.1.html