Carrés magiques 3×3 de carrés

Lucien PECH - lucien.pech@ens.fr

1 décembre 2006

Un carré magique $n \times n$ est un n^2-uplet d'entiers distincts, disposés sous forme d'un carré, de telle sorte que les sommes des lignes, des colonnes et des deux grandes diagonales soient toutes égales.

On ne connaît pas de carré magique 3×3 composé uniquement d'entiers carrés distincts, mais il n'a pas été prouvé qu'un tel carré n'existait pas. On présentera donc les démarches envisageables pour chercher un tel carré, ainsi qu'un algorithme efficace permettant de vérifier qu'il n'existe pas de tel carré dont la valeur de la case centrale n'excède pas une valeur donnée. Cet algorithme s'appuie sur les propriétés des sommes de deux carrés.

Comme dans l'article de Duncan Buell [2], on ne cherchera non pas un carré magique, mais seulement un "sablier" magique :

$a-b$	$a-c$	$a-d$
	a	
$a+d$	$a+c$	$a+b$

1 Principe - Somme de deux carrés

Quelques résultats mathématiques

Pour simplifier les notations, on notera $\mathcal{P}_1$ l'ensemble de entiers positifs premiers de la forme $4 \cdot k + 1$ (k entier), et $\mathcal{P}_3$ ceux de la forme $4 \cdot k + 3$.

On va utiliser les résultats classiques suivants :

Lemme 1 :
L'équation $x^2 + y^2 = z^2$, avec $x,\ y,\ z \in \mathbb{N}^*$ premiers entre eux dans leur ensemble, a pour solution (à interversion de x et de y près) $\{(x, y, z) = (2 \cdot m \cdot n,\ m^2 - n^2,\ m^2 + n^2),\ (m,\ n) \in (\mathbb{N}^*)^2,\ m > n\}$.

Théorème 1 :
Un entier $n > 0$ est la somme de deux carrés si et seulement si tous les facteurs premiers de n dans $\mathcal{P}_3$ ont des exposants pairs dans la décomposition de n en produit de facteurs premiers.

Somme de deux carrés

On se place dans le cas où tous les éléments du sablier sont des entiers carrés distincts, premiers entre eux dans leur ensemble.

Si on pose $x^2 = a - b$, $y^2 = a + b$ et $A^2 = a$, on a $2 \cdot A^2 = x^2 + y^2$. En posant le changement d'inconnues $x = u - v$ et $y = u + v$, cette équation se ramène à l'équation $A^2 = u^2 + v^2$.

D'après le lemme 1, les solutions de cette équation sont de la forme $(A,\ u,\ v) = (m^2 + n^2,\ 2 \cdot m \cdot n,\ m^2 - n^2)$, où m et n sont des entiers positifs.

On a alors $2 \cdot A^2 = (u + v)^2 + (u - v)^2$.

Donc

$$\begin{aligned} b &= a - x^2 \\ &= y^2 - a \\ &= (u+v)^2 - A^2 \\ &= (2 \cdot m \cdot n + m^2 - n^2)^2 - (m^2 + n^2)^2 \\ &= 4 \cdot m \cdot n \cdot (m^2 - n^2) \end{aligned}$$

Il faut donc que A soit somme de deux carrés d'au moins trois façons différentes.

Supposons que l'on ait $A = m^2 + n^2 = r^2 + s^2 = u^2 + v^2$. On a alors :

$$\begin{aligned} b &= 4 \cdot m \cdot n \cdot (m^2 - n^2) \\ c &= 4 \cdot r \cdot s \cdot (r^2 - s^2) \\ d &= 4 \cdot u \cdot v \cdot (u^2 - v^2) \end{aligned}$$

à permutation de m et n, r et s, u et v près (b, c et d ne sont pas tous positifs).

Si a est pair, alors $4 \mid a$, car a est un carré. Or $4 \mid b$, $4 \mid c$ et $4 \mid d$, ce qui contredit que les éléments du carré sont premiers entre eux dans leur ensemble. Dans la suite, on supposera toujours que a **est impair**, et est donc somme de deux carrés de parités différentes.

Equation à vérifier

La condition pour que les sommes des lignes horizontales soient égales à $3 \cdot a$ est :

$$b + c + d = 0 \quad (\mathcal{E})$$

Algorithmes

Le principe des deux algorithmes que nous allons présenter est le suivant : étant donné un entier positif A_{max}, pour tout entier positif $A \leqslant A_{max}$ qui puisse s'écrire comme la somme de deux carrés d'au moins trois façons, on calcule pour tout $\{m,\ n,\ r,\ s,\ u,\ v\} \subset \mathbb{N}$ tel que $A = m^2 + n^2 = r^2 + s^2 = u^2 + v^2$, les valeurs de b, c et d données par les relations précédentes, et on teste si l'équation $(\mathcal{E})$ est vérifiée.

2 Tester toutes les valeurs de b, c et d

Principe

Soient a et M deux entiers positifs. On dispose d'une liste de N paires $\{m,\ n\}$ distinctes d'entiers, $N \geqslant 3$, telles que $a = m^2 + n^2$. Les valeurs de b, c et d pouvant être grandes et ne pouvant donc pas être stockées dans une variable de 64 bits, on teste l'égalité $\mathcal{E}$ uniquement modulo M.

On trie les valeurs obtenues pour b (respectivement c et d) pour chaque paire $\{m,\ n\}$. Pour un b fixé, pour toutes les valeurs de c, on peut obtenir la valeur de d la plus proche de $-b-c$ modulo M en temps constant, en gardant en mémoire la position de d pour la valeur précédente de c, et en la décalant au fur et à mesure.

Le tri coûte $O(N \cdot \log(N))$, les deux boucles $O(N^2)$. La complexité totale est alors $O(N^2)$.

On supposera par la suite que l'on dispose d'une fonction `tester_tous_couples` effectuant ces opérations et affichant les éventuels résultats.

3 Algorithme de Buell

Principe

On teste toutes les valeurs entières possibles de u et v telles que $A = u^2 + v^2 \leqslant A_{max}$. On retient, pour chaque valeur possible de A les paires $\{u,\ v\}$ telles que $A = u^2 + v^2$.

Algorithme

On sait que A doit être impair ; on peut donc supposer u pair (que l'on appellera dorénavant p) et v impair (qui devient i).

La mémoire disponible étant insuffisante pour enregistrer les listes de couples $(i,\ p)$ pour toutes les valeurs de A lorsque A_{max} est grand, on fixe un entier I (que l'on choisit comme le plus grand entier possible pour lequel il soit possible d'enregistrer $I/4$ listes de couples $(i,\ p)$), et on applique l'algorithme qui précède aux entiers $A = i^2 + p^2 \in [k \cdot I,\ (k+1) \cdot I]$ pour toutes les valeurs de k entières telles que $k \cdot I < A_{max}$.

Algorithme 1 Algorithme de Buell - fonction `buell`

Entrées: $A_{max} \in \mathbb{N}$, $I \in \mathbb{N}$

```
 1. k ← 0
 2. tant que k·I < A_max faire
 3.     liste[0..I/4] ← ∅
 4.     i ← 1
 5.     tant que i² < min((k+1)·I, A_max) faire
 6.         p ← ⌊√max(k·I − i², 0)⌋
 7.         si p impair alors
 8.             p ← p + 1
 9.         fin si
10.         tant que (s ← i² + p²) ⩽ min(A_max, (k+1)·I − 1) faire
11.             liste[(s − 1)/4] ← (i, p)::liste[(s − 1)/4]
12.             p ← p + 2
13.         fin tant que
14.         i ← i + 2
15.     fin tant que
16.     pour j ← 0 croissant jusqu'à I/4 faire
17.         si taille(liste[j]) ⩾ 3 alors
18.             tester_tous_couples(liste[j])
19.         fin si
20.     fin pour
21.     k ← k + 1
22. fin tant que
```

Complexité et temps d'exécution :

Les lignes 11 et 12 sont exécutées une fois pour chaque valeur de i impaire et p paire telle que $i^2 + p^2 \leqslant A_{max}$, soit $A_{max}/4$ fois. Les lignes 17 à 19 sont également exécutées $A_{max}/4$ fois.

Pour évaluer le coût total des appels à `tester_tous_couples` (dont la complexité est quadratique en la taille de la liste fournie), on mesure, ligne 18, la moyenne quadratique de **taille**(liste[j]), pour $j \in [1..A_{max}/4]$. Pour $A_{max} = 10^{10}$, cette moyenne est égale à 3,2 et pour $j \in [1000 \cdot 10^{10},\ 1001 \cdot 10^{10}]$, elle est de 4,2. On prendra toujours $A_{max} \leqslant 10^{13}$; dans ce domaine on pourra donc considérer que les lignes 16 à 20 coûtent $O(A_{max})$.

La boucle **tant que** des lignes 2 à 22 est exécutée A_{max}/I fois. La boucle **tant que** des lignes 5 à 15 est exécutée au plus $\sqrt{A_{max}}$ fois. Le lignes 6 à 14 sont donc exécutées $O(A_{max}^{3/2}/I)$ fois.

Le coût total est donc $O(A_{max} + A_{max}^{3/2}/I)$. Pour I du même ordre de grandeur que $\sqrt{A_{max}}$, le coût est donc linéaire en A_{max}. Mais la mémoire étant limitée dans la pratique, c'est le terme en $A_{max}^{3/2}/I$ qui domine pour de grandes valeurs de A_{max}.

On donne à titre indicatif le temps d'exécution de notre programme pour quelques valeurs de A_{max}, ainsi que le quotient temps / A_{max} à une constante multiplicative près (processeur 64 bits cadencé à 2 GHz) :

A_{max}	Temps d'exécution (en secondes)	Temps / A_{max}
10^9	50,85	1
10^{10}	535,62	1,053
10^{11}	5.958,70	1,172
$2 \cdot 10^{12}$	142.617,20	1,402

Remarques, avantages et inconvénients de la méthode :

En 1998, Duncan Buell a ainsi vérifié qu'il n'existait pas de "sablier" magique (et donc *a fortiori* pas de carré magique) d'entiers carrés qui vérifie l'équation $\mathcal{E}$, pour $A_{max} = 5 \cdot 10^{12}$. Contrairement à nous, il a également testé les sommes d'entiers carrés impairs. Le temps d'exécution de son programme a été de près d'un an sur une station Silicon Graphics.

Cette méthode présente l'avantage d'être assez simple à implémenter, et de facilement se paralléliser : on peut travailler sur n'importe quel intervalle.

4 Amélioration

Principe - Réduire le nombre de valeurs de A possibles

On suppose que a, b et c sont premiers entre eux dans leur ensemble.

Soit $A = m^2 + n^2$. Si $p \in \mathcal{P}_3$ est tel que $p^2 \mid A$, alors $p \mid m$ et $p \mid n$, donc $p \mid b$ et $p \mid c$. On considèrera donc que tous les facteurs premiers de A sont dans $\mathcal{P}_1$.

Nous allons donc construire récursivement les valeurs possibles de A comme produits d'éléments de $\mathcal{P}_1$.

$p \in \mathcal{P}_1$ s'écrit de manière unique comme la somme de deux entiers positifs. Un produit de deux somme de carrés s'écrit de deux manières comme la somme de deux carrés :

$$\begin{aligned} (a^2+b^2)\cdot(c^2+d^2) &= (a\cdot d + b\cdot c)^2 + (a\cdot c - b\cdot d)^2 \qquad (\mathcal{S}) \\ &= (a\cdot d - b\cdot c)^2 + (a\cdot c + b\cdot d)^2 \qquad (\mathcal{S}') \end{aligned}$$

Pratique

Soit $A \leqslant A_{max}$ entier qui s'écrit d'au moins trois manières comme la somme de deux carrés et qui n'a pas de diviseur dans $\mathcal{P}_3$. A admet alors au moins trois diviseurs. Si $p \in \mathcal{P}_1$ est tel que $p^2 \mid A$, alors $p \leqslant \sqrt{A/5}$. Soit $p_{max} = \sqrt{A_{max}/5}$.

Soient $5 = p_1 < p_2 < ... < p_n$ les éléments de $\mathcal{P}_1$ inférieurs à p_{max}. De même que dans l'algorithme de Buell, on calcule leur unique décomposition comme somme de carrés. Pour tout k entier, $k \in [1,\ n]$, on calcule avec $(\mathcal{S})$ et $(\mathcal{S}')$, pour tout α entier tel que $p_k^\alpha \leqslant A$, les décompositions de p_k^α comme somme de deux carrés.

Pour tout $(\alpha_1,\ ...,\ \alpha_n) \in \mathbb{N}^n$ tel que $A = \prod_{k=1}^{n} p_k^{\alpha_k} \leqslant A_{max}$, on calcule avec $(\mathcal{S})$ et $(\mathcal{S}')$ la liste des décompositions de A comme somme de deux carrés. On applique alors `tester_tous_couples` à la liste ainsi obtenue.

Si A est un entier, $A \leqslant A_{max}$, et si $p \in \mathcal{P}_1$, $p > p_{max}$ tel que $p \mid A$, alors $p^2 \nmid A$. $A/p < \sqrt{5 \cdot A_{max}}$. On retient donc les décompositions comme somme de deux carrés des entiers inférieurs à $\sqrt{5 \cdot A_{max}}$ lors de la première étape de l'algorithme. Comme pour l'algorithme de Buell, pour tous les m impairs et n pairs tels que $p_{max} < p = m^2 + n^2 \leqslant A_{max}/25$, pour tous les $s \leqslant A_{max}/p < \sqrt{5 \cdot A_{max}}$ (dont on a retenu la liste des décompositions comme somme de deux carrés), on calcule les décompositions de $p \cdot s$ avec $(\mathcal{S})$ et $(\mathcal{S}')$ comme somme de deux carrés, et on applique `tester_tous_couples` à la liste ainsi obtenue.

Algorithme

Nous ne détaillerons pas toutes les étapes de l'algorithme, mais seulement les deux parties essentielles.

On suppose que l'on dispose d'un tableau `premier`$[1..n]$ qui contient les entiers p_1, ..., p_n, et d'un tableau de listes de couples `sommes_carres`$[1..n][1..\lfloor \log_5(A_{max}) \rfloor]$. `sommes_carres`$[k][\alpha]$ contient la liste des couples $(i,\ p)$ tels que $p_k^\alpha = i^2 + p^2$.

La première fonction : `tester_produits`, calcule tous les $A = \prod_{k=1}^{n} p_k^{\alpha_k} \leqslant A_{max}$. Elle utilise une pile, initialement vide, dont les éléments sont des couples $(k,\ \alpha_k)$, où k est l'indice d'un entier p_k, et α_k est sa puissance dans le produit de A. On suppose que l'on dispose d'une fonction `construire_sommes`, qui à partir des éléments déjà empilés de produit A, calcule la liste des couples $(i,\ p)$ tels que $A = i^2 + p^2$, et appelle ensuite la fonction `tester_tous_couples` sur cette liste.

Algorithme 2 Fonction `tester_produits`

Entrées: $k \in [1,\ n]$, $P \in \mathbb{N}$, nb_sommes $\in \mathbb{N}$

```
1.  si nb_sommes ⩾ 3 alors
2.      construire_sommes()
3.  fin si
4.  k_p ← k
5.  tant que k_p ⩽ k et P·premier[k_p] ⩽ A_max faire
6.      α ← 1
7.      tant que P·premier[k_p]^α ⩽ A_max faire
8.          empiler((k_p, α))
9.          tester_produits(k_p + 1, P·premier[k_p]^α,
            nb_sommes + taille(sommes_carres[k_p][α])
10.         dépiler()
11.         α ← α + 1
12.     fin tant que
13.     k_p ← k_p + 1
14. fin tant que
```

La fonction `tester_gros_facteurs` gère les valeurs A admettant des diviseurs premiers supérieurs à p_{max}.

Complexité et temps d'exécution :

On ne calculera que la complexité de la fonction `tester_gros_facteurs`, celle-ci coûtant expérimentalement environ les trois quarts du temps d'exécution de notre programme.

Cette fonction est assez proche de la fonction `buell`. Les couples $(i_A,\ p_A)$ et $(i'_A,\ p'_A)$ ajoutés à `sommes_carrés_bis` lignes 27 et 29 sont tous distincts ; ces lignes sont donc excutées strictement

Algorithme 3 Fonction `tester_gros_facteurs`

Entrées: $A_{max} \in \mathbb{N}$

1. liste_facteurs $\leftarrow \emptyset$
2. $k \leftarrow 4 \cdot \left\lfloor \frac{\sqrt{5 \cdot A_{max}} - 1}{4} \right\rfloor$
3. **tant que** $k \geqslant 1$ **faire**
4. **si taille**(sommes_carrés[k]) $\geqslant 2$ **alors**
5. liste_facteurs $\leftarrow k$ ::liste_facteurs
6. **fin si**
7. $k \leftarrow k - 1$
8. **fin tant que**
9. $i_s \leftarrow 1$
10. **tant que** $i_s^2 \leqslant \sqrt{A_{max}}/5$ **faire**
11. $p_s \leftarrow \left\lfloor \sqrt{max(\sqrt{A_{max}}/5 - i_s^2,\ 0)} \right\rfloor$
12. **si** p_s **impair alors**
13. $p_s \leftarrow p_s + 1$
14. **fin si**
15. **tant que** $(s \leftarrow i_s^2 + p_s^2) \leqslant A_{max}$ **faire**
16. **si est_probablement_premier**(s) **alors**
17. liste_facteurs_local $\leftarrow$ liste_facteurs
18. **tant que** $s \times (k \leftarrow$ **tête**(liste_facteurs_local)) $\leqslant A_{max}$ **faire**
19. l $\leftarrow$ sommes_carrés[k]
20. somme_carrés_bis $\leftarrow \emptyset$
21. **tant que** l $\neq \emptyset$ **faire**
22. $(i_{4 \cdot k+1},\ p_{4 \cdot k+1}) \leftarrow$ **tête**(l)
23. $i_A \leftarrow |i_s \cdot i_{4 \cdot k+1} - p_s \cdot p_{4 \cdot k+1}|$
24. $p_A \leftarrow i_s \cdot p_{4 \cdot k+1} + p_s \cdot i_{4 \cdot k+1}$
25. $i'_A \leftarrow i_s \cdot i_{4 \cdot k+1} + p_s \cdot p_{4 \cdot k+1}$
26. $p'_A \leftarrow |i_s \cdot p_{4 \cdot k+1} - p_s \cdot i_{4 \cdot k+1}|$
27. somme_carrés_bis $\leftarrow (i_A,\ p_A)$::sommes_carrés_bis
28. **si** $i_A \neq i'_A$ **alors**
29. somme_carrés_bis $\leftarrow (i'_A,\ p'_A)$::sommes_carrés_bis
30. **fin si**
31. l $\leftarrow$ **queue**(l)
32. **fin tant que**
33. tester_tous_couples(somme_carrés_bis)
34. liste_facteurs_local $\leftarrow$ **queue**(liste_facteurs_local)
35. **fin tant que**
36. **fin si**
37. $p_s \leftarrow p_s + 2$
38. **fin tant que**
39. $i_s \leftarrow i_s + 2$
40. **fin tant que**

moins de $A_{max}/4$ fois. Donc si, de même que pour `buell`, on considère que `tester_tous_couples` coûte un temps constant, le coût total est en $O(A_{max})$.

Pour la fonction **est_probablement_premier**, on utilise la fonction de la bibliothèque `gmp` (bibliothèque très performante permettant des opérations sur des nombres de taille quelconque, *c.f.* [4]) ; les valeurs de s étant petites (elles peuvent être stockées dans une variable de 64 bits), et étant donné qu'une erreur de cette fonction ne contribue pas à une erreur de notre algorithme, et que l'on peut donc utiliser cette fonction avec peu d'essais (5 dans notre cas), on considèrera que le temps d'exécution de cette fonction n'est pas plus important que le temps d'exécution des lignes 18 à 36 (expérimentalement, les temps d'exécution sont approximativement les mêmes).

Sous ces différentes hypothèses, le coût total est donc $O(A_{max})$.

On donne ici les temps d'exécution de la fonction `tester_produits`, du nombre de solutions trouvées par cette dernière, et du temps total d'exécution de notre programme pour différentes valeurs de A_{max}.

A_{max}(modulo)	Temps `tester_produits` (s)	Solutions	Total (s)	Temps / A_{max}
10^9 (2^{32})	8,07	6/10	32,39	1
10^{10} (2^{35})	90,59	4/6	332,32	1,026
10^{11}(2^{39})	1.022,20	6/10	3.518,01	1,086
10^{12}(2^{42})	11.484,39	6/9	37.671,46	1,163
$5 \cdot 10^{12}$ (2^{44})	60.912,43	6/8	195.415,61	1,207

On remarque donc que le temps d'exécution de la fonction `tester_produits` représente environ un quart du temps d'exécution de notre programme, mais qu'elle produit la majorité des résultats. En effet, plus A s'écrit de nombreuses manières comme la somme de 2 carrés, *i.e.* plus A a de diviseurs, et plus il y a de chances que l'équation $\mathcal{E}$ soit vérifiée.

On peut donc envisager de n'exécuter que la fonction `tester_produits` ; on n'est alors plus assuré de tester toutes les valeurs de A inférieures à A_{max}, mais on cherche parmi les entiers qui ont le plus de chances d'être solution de $\mathcal{E}$. On peut ainsi rapidement trouver de "grandes" (*i.e.* modulo une grande valeur) solutions de $\mathcal{E}$.

À titre d'exemple, pour les valeurs suivantes, le "sablier" d'entiers carrés obtenu est magique modulo 2^{52} (la plus grande solution obtenue par Duncan Buell est magique modulo 2^{46}) :

$$\begin{aligned} m &= 6881928 \\ n &= 4357501 \\ r &= 1580988 \\ s &= 7990571 \\ u &= 216213 \\ v &= 8142604 \end{aligned}$$

On donne également les solutions impaires modulo une puissance de 2 supérieure à 45 de $\mathcal{E}$, pour $A_{max} = 10^{13}$:

A	modulo	m	n	r	s	u	v
2726033914369	2^{46}	1084080	1245313	1643860	154137	1613863	348540
7886415129265	2^{47}	2776608	420551	2150073	1806544	1199431	2539248
8100428953889	2^{45}	2312020	1659817	2346808	1610255	2179817	1829980
9082490568241	2^{46}	723720	2925529	2015296	2240775	2839079	1011000

Références

[1] Christian Boyer, A search for 3x3 magic squares having more than six square integers among their nine distinct integers, preprint, 2004

[2] Duncan A. Buell, A search for a magic hourglass, preprint, 1999

[3] Daniel Duverney, Théorie des nombres, Dunod, 1998

[4] GNU multiple precision arithmetic library, http ://www.swox.com/gmp/

[5] Xavier Gourdon, Algèbre, Ellipses, 1994

[6] Yves Hellegouarch, Invitation aux mathématiques de Fermat-Wiles, Dunod, 1997