HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG

Fakultät Informatik

Das Luzifer-Rätsel

Prof. Dr. Hartmut Plesske
Wintersemester 2008/09

von

Max Nagl
nagl@fh-konstanz.de

Inhaltsverzeichnis

1. Einleitung

Diese Ausarbeitung beschäftigt sich mit einer experimentalmathematischen Lösung des Luzifer-Rätsels. Die Aufgabenstellung dieses Rätsels wurde der Wikipedia, einer Online-Enzyklopädie, entnommen:

Es kursieren verschiedene Fassungen des Rätsels, die inhaltlich identisch sind und sich lediglich im textlichen Rahmen unterscheiden. Eine populäre Fassung, die zur Bezeichnung „Luzifer-Rätsel“ führte, lautet in etwa folgendermaßen:

Die berühmten Mathematiker Carl Friedrich Gauß und Leonhard Euler landen nach ihrem Tod in der Hölle. Luzifer verspricht ihnen die Freiheit, wenn sie die beiden ganzen Zahlen zwischen 1 und 100 erraten, die er sich ausgedacht hat. Er nennt Gauß das Produkt und Euler die Summe der beiden Zahlen; darauf entwickelt sich zwischen den Mathematikern folgender Dialog:

Gauß: „Ich kenne die beiden Zahlen nicht.“

Euler: „Das war mir klar.“

Gauß: „Jetzt kenne ich die beiden Zahlen.“

Euler: „Dann kenne ich sie jetzt auch.“

Unabhängig von der Frage, ob Gauß und Euler aus der Hölle entkommen, lautet die Aufgabe, allein aus diesen Angaben die beiden Ausgangszahlen zu ermitteln.[Wikipedia, 2008]

Es wird vorausgesetzt, dass sich der Leser bereits mit dem Rätsel beschäftigt hat und der grundlegende Lösungsansatz verstanden wurde.

2. Der Algorithmus

Der Algorithmus besteht aus 4 Schritten, welche in diesem Abschnitt erläutert werden.

Ziel dieses Algorithmuses ist es unter anderem, den Zahlenbereich variabel zu definieren. Aus diesem Grund liegt Bereich der zu erratenden Zahlen im Bereich von `(2,N)`, wobei `N` eine beliebige ganze Zahl größer als 2 darstellt.

2.1. Faktorenpaare Teil 1

Für die Lösung des Rätsels ist es von entscheidender Bedeutung zu wissen, wie viele Faktorenpaare für eine gegebene Zahl `x` existieren.

Hierzu wird ein Feld `produkte` mit der Größe N^2 erstellt. Anschließend werden zwei verschachtelte Schleifen durchlaufen. In der äußeren Schleife wird die Variable `a` im Bereich `(2,N)` durchlaufen. Im Inneren durchläuft die Variable `b` den Bereich `(a,N)`. Für jedes Wertepaar wird nun die Variable `produkte[a*b]` um 1 dekrementiert. Nach dem die Schleifen durchlaufen wurden, steht in dem Feld `produkte` wie viele Faktorenpaare für jede Zahl existieren.

Warum die Werte in dem Feld `produkte` dekrementiert und nicht inkrementiert werden, wird später erklärt.

Der dazugehörige Quelltext ist hier abgedruckt:

```
1 for (a = 2; a <= N; a++) for (b = a; b <= N; b++)
2     produkte[a*b]--;
```

Die Laufzeit und der Speicherverbrauch dieses Algorithmuses nimmt quadratisch mit `N` zu.

2.2. Mögliche Summen Teil 1

Um herauszufinden welche Summen möglich sind, muss überprüft werden, für welche Summen kein einziges Summandenpaar a, b existiert, für welches gilt: Das Produkt aus a und b darf nur ein einziges Faktorenpaar besitzen.

Dazu wird eine weiteres Feld `summe` mit der Größe `2*N` benötigt. Auch hier werden zwei verschachtelte Schleifen wie bei den Faktorenpaaren durchlaufen. Um die Effizienz zu steigern, wird hier allerdings ein anderes Iterationsverfahren verwendet. Die äußere Schleife zählt für `b` von `2` bis `max+2`. Die Innere zählt von `2` bis `b/2`. `b` stellt hier die Summe und `a` den ersten Summand dar. Der zweite Summand berechnet sich mit Hilfe der Formel `(b-a)`.

Jedes Mal, wenn gilt `produkte[a*(b-a)] == -1`, wird `summe[b]` um eins inkrementiert. Im Anschluss steht im Feld `summe` wie viele Summandenpaare mit der oben genannten Bedingung für jede Summe existieren. Ist `summe[x] = 0` so ist dies eine gültige Summe.

Hier der Quelltext:

```
1 for (b = 4; b <= 2*N; b++) for (a = 2; a*2 < b; a++)
2     if (produkte[(b-a)*a] == -1) { summe[b]=1; break; }
```

Die Laufzeit dieses Algorithmuses nimmt im schlimmsten Fall quadratisch mit `N` zu. Der Speicherbedarf steigt linear an.

2.3. Ein kleiner Optimierungsschritt

Da die möglichen Zahlenpaare durch die maximal mögliche Summe begrenzt sind, wird nun die größte mögliche Summe gesucht und als `MAX` gespeichert. Ab sofort muss in den Schleifen nicht mehr bis `N` sondern nur noch bis `MAX` iteriert werden.

Dies ist ein optionaler Zwischenschritt. In der Nummerierung der Schritte spielt er aus diesem Grund keine Rolle und wird nicht berücksichtigt.

Hier der Quelltext:

```
1 for (a = N; a > 2; a--) if (summe[a]==0) {max=(short) (a-2);break;}
```

Die Laufzeit dieses Algorithmuses nimmt linear mit `N` zu. Es entsteht ein minimaler zusätzlicher Speicherbedarf für eine Variable.

2.4. Faktorenpaare Teil 2

Nun muss herausgeunden werden, für welche Produkte die folgende Eigenschaft zustrifft: Die Summe von exakt einem Faktorenpaar muss eine gültige Summe sein.

Auch hier wird für jedes Produkt in einem Feld gespeichert, ob es dieses Kriterien erfüllt. Um den Speicherverbrauch aber nicht weiter zu erhöhen, wird das Feld `produkte` wiederverwendet. Aus diesem Grund wurden zuvor die Werte dekrementiert. Alle Felder mit einem Wert von kleiner Null werden als Null betrachtet.

Nach diesem Schritt sollen in dem Feld `produkte` folgende Werte stehen:

- <=0 Falls kein Faktorenpaar existiert.
- 1 Falls exakt ein Faktorenpaar mit einer gültigen Summe existiert.
- 2 Falls mehrere Faktorenpaare mit einer gültigen Summe existieren.

Es sind also alle Produkte `X` gültig, für die gilt: `produkte[x] = 1`.

Erneut werden zwei verschachtelte Schleifen durchlaufen. Allerdings muss die innere Schleifen nur durchlaufen werden, falls es sich bei `b` um eine gültige Summe handelt. Die Werte im Feld `produkte` werden entsprechend angepasst.

Hier der entsprechende Quelltext:

```
1 for (b = 4; b < max; b++) if (summe[b]==0) for (a = 2; a*2 < b; a++)
2     produkte[(b-a)*a] = produkte[(b-a)*a]<=0?(byte)1:2;
```

Die Laufzeit dieses Algorithmuses nimmt quadratisch mit `N` zu. Es entsteht kein zusätzlicher Speicherbedarf.

2.5. Mögliche Summen Teil 2

Im letzten Schritt müssen Summen gesucht werden, bei denen nur ein einziges Summandenpaar existiert, welches die zweite Faktorenpaar-Regel erfüllt.

Zum letzten Mal werden zwei Schleifen durchlaufen. Die äußere Schleife zählt für `b` von `2` bis `max+2`. Die Innere zählt von `2` bis `b/2`. `b` stellt hier die Summe und `a` den ersten Summand dar. Der zweite Summand berechnet sich mit Hilfe der Formel `(b-a)`. Wird nun während dem Durchlaufen der inneren Schleife zum ersten Mal ein Summandenpaar mit gültigem Produkt gefunden, so wird der erste Summand in der Variable `s` gespeichert. Wird ein zweites Summandenpaar gefunden, wird `s` auf Null zurückgesetzt und die innere

Schleif abgebrochen. Wird die innere Schleife vollständig durchlaufen und enthält `s` einen Wert ungleich 0, so wird eine mögliche Lösung gefunden und ausgegeben.

Hier der entsprechende Quelltext:

```
for (b = 2; b < max+2; b++)
    if (summe[b] == (s=0)) {
        for (a = 2; a*2 < b; a++)
            if (produkte[a*(b-a)]==1)
                if ((s=s==0?a:0)==0) break;
        if (s != 0)
            System.out.println("Lösung "+n+":"+" a = "+s+", b = "+(b-s));
    }
```

Die Laufzeit dieses Algorithmuses nimmt quadratisch mit `N` zu. Es entsteht ein minimaler zusätzlicher Speicherbedarf für eine Variable.

2.6. Der vollständige Algorithmus

Hier noch einmal der vollständige Algorithmus:

```
for (a = 2; a <= N; a++) for (b = a; b <= N; b++)
    produkte[a*b]--;

for (b = 4; b <= 2*N; b++) for (a = 2; a*2 < b; a++)
    if (produkte[(b-a)*a] == -1) { summe[b]=1; break; }

for (a = N; a > 2; a--) if (summe[a]==0) {max=(short) (a-2);break;}

for (b = 4; b < max; b++) if (summe[b]==0) for (a = 2; a*2 < b; a++)
    produkte[(b-a)*a] = produkte[(b-a)*a]<=0?(byte)1:2;

for (b = 2; b < max+2; b++)
    if (summe[b] == (s=0)) {
        for (a = 2; a*2 < b; a++)
            if (produkte[a*(b-a)]==1)
                if ((s=s==0?a:0)==0) break;
        if (s != 0)
            System.out.println("Lösung "+n+":"+" a = "+s+", b = "+(b-s));
    }
```

2.7. Laufzeit und Speicherbedarf

Wie oben bereits erwähnt, erhöht sich die Laufzeit der einzelnen Teile jeweils quadratisch. Aus diesem Grund ist auch für die Gesammtlaufzeit ein quadratisches Verhalten zu erwarten.

Es wurden mehrere Benchmarks durchgeführt. Dazu wurde der Code mehrmals durchlaufen und der durchschnittliche Zeitbedarf berechnet. Die Ergebnisse sind in Tabelle 2.1 zu sehen. Auf Grund des extrem hohen Speicherbedarfs konnten keine N größer als 31999 getestet werden.

N	Laufzeit
99	$27\mu s$
999	$3,7ms$
9999	$2,2s$
31999	$30s$
31999*	$27s$

Tabelle 2.1.: Laufzeit bei verschiedenen N

Bei den Tests ist ein mehr als quadratisches Verhalten der Laufzeit zu erkennen. Der Grund hierfür liegt vermutlich daran, dass bei `N = 99` der gesamte verwendete Speicher im CPU-Cache gehalten werden kann. Bei größeren N ist dies nicht der Fall. Die Werte müssen häufig aus dem erheblich langsameren Hauptspeicher nachgeladen werden.

Als besonders erstaunlich sind die letzten beiden Ergebnisse der Tabelle 2.1. In der vorletzten Zeile wurde der Algorithmus mit `N = 31999` nur ein Mal durchlaufen und benötigte dabei 30 Sekunden.

In der letzten Zeile wurde der Algorithmus mit `N = 99` einhundert mal durchlaufen. Anschließend noch ein mal mit `N = 31999`. Hierdurch hat sich die gesamte Laufzeit um ca. 10% verringert, obwohl mehr berechnet werden musste. Dies liegt vermutlich daran, dass die JVM den Code zur Laufzeit zu optimieren versucht. Dies gelingt um so besser, um so häufiger ein Algorithmus durchlaufen wurde.

Auf Grund des Laufzeitverhaltens wäre es durchaus denkbar, denn Algorithmus für `N = 999999` laufen zu lassen. Das Problem ist allerdings der quadratisch ansteigende Speicherbedarf. Bei `N = 31999` wird bereits circa 1 GiB Speicher benötigt. Bei `N = 999999` würde etwa 1 TiB Speicher benötigt. Aus diesem Grund wurde im nächsten Abschnitt versucht, den Speicher zu partitionieren.

2.8. Partitionierung des Speichers

Um den Speicherbedarf des Programms zu vermindern, wurde der benötigte Speicher in gleich große Teile aufgeteilt, wobei immer nur ein Teil zu gleich benötigt wird. Um eine möglichst effiziente Datenstuktur zu erhalten wird die Größe der Teile auf die halbe Größe des CPU-Caches gesetzt.

Um den Algorithmus zu partitionieren, muss `produkte` im ersten Teil aufgeteilt werden. Da die Partitionen im zweiten Teil benötigt werden, muss dieser Teil ebenfalls Teil der Partitionierung sein. Im nächsten Teil ist es allerdings notwendig auf das gesamte Feld `summe` zugreifen zu können. Daher muss die Partitionierung vor dem 3. Teil wieder zusammengefasst werden. Teil 3 und 4 müssen ebenfalls wieder zusammen in einer Partitionierung stecken.

Für diesen Algorithmus werden zwei weitere Felder der Größe `2*N` benötigt. Dafür ist die Grösse des Feldes `produkte` nicht mehr `N*N` sonder, nur noch `N`. Somit steigen alle verwendeten Datenstrukturen maximal linear mit `N` an.

Die Laufzeit des partitionierten Algorithmuses ist weiterhin quadratisch, wobei hier auf einen exakten Beweis verzichtet wird. Die Laufzeit bei gleichem N entspricht etwa der zehnfachen Zeit des unpartitionierten Algorithmuses.

Der Quelltext ist erheblich länger als bei der vorherigen Version und ist deshalb hier nicht abgedruckt, sonder nur auf der beiliegenden CD enthalten.

2.9. Ergebnisse

Die Ergebnisse für verschiedene `N` sind im Anhang zu finden.

Auffällig ist, dass eine der beiden Zahlen meistens eine Zweierpotenz, die andere eine Primzahl ist. Eine mögliche Begründung hierfür liegt in der goldbachschen Vermutung. Diese besagt, dass jede gerade Zahl größer als Zwei als Summe zweier Primzahlen geschrieben werden kann. Nach der algebraischen Lösung sollten deshalb nur Zahlenpaare bestehend aus einer Zweierpotenz und einer Primzahl vorkommen. Dies ist aber nicht immer der Fall. Wie in den Lösungen ersichtlich, kommen immer wieder andere Zahlenpaare vor. Diese verschwinden allerding bei steigendem `N` wieder. Ein Beispiel hierfür sind die Zahlen `416` und `827` bei `N = 49.999`. Erhöht man `N` auf `99.999`, so verschwindet das Zahlenpaar wieder.

Als Begründung wird nun an einem Beispiel gezeigt, warum die goldbachschen Vermutung bei diesem Problem nicht verwendet werden kann. So kann die Zahl 188 als Summe der folgenden Primzahlen geschrieben werden:

- 7 + 181
- 31 + 157
- 37 + 151
- 61 + 127
- 79 + 109

Diese Zahlenpaare sind allerdings bei `N = 99` nicht erlaubt, da der zweite Summand immer größer ist als einhundert. Wird `N` vergrößert, so sind diese Zahlenpaare wieder erlaubt. Aus diesem Grund kommen Zahlenpaare, die keine Zweierpotenz enthalten, gelegentlich vor und verschwinden später wieder.

A. Ergebnisse

A.1. N = 100

Lösung 1: a = 4, b = 13

A.2. N = 1.000

Lösung 1: a = 4, b = 13
Lösung 2: a = 4, b = 61

A.3. N = 5.000

Lösung 1: a = 4, b = 13
Lösung 2: a = 4, b = 61
Lösung 3: a = 16, b = 73
Lösung 4: a = 16, b = 111
Lösung 5: a = 64, b = 73
Lösung 6: a = 67, b = 82
Lösung 7: a = 32, b = 131
Lösung 8: a = 4, b = 229
Lösung 9: a = 32, b = 311
Lösung 10: a = 64, b = 309

A.4. N = 10.000

Lösung 1: a = 4, b = 13
Lösung 2: a = 4, b = 61
Lösung 3: a = 16, b = 73
Lösung 4: a = 16, b = 111
Lösung 5: a = 64, b = 73
Lösung 6: a = 32, b = 131
Lösung 7: a = 16, b = 163
Lösung 8: a = 4, b = 181
Lösung 9: a = 64, b = 127
Lösung 10: a = 4, b = 229
Lösung 11: a = 8, b = 239
Lösung 12: a = 32, b = 311
Lösung 13: a = 64, b = 309

A.5. N = 32.000

Lösung 1: a = 4, b = 13
Lösung 2: a = 4, b = 61
Lösung 3: a = 16, b = 73
Lösung 4: a = 16, b = 111
Lösung 5: a = 64, b = 73
Lösung 6: a = 32, b = 131
Lösung 7: a = 16, b = 163
Lösung 8: a = 4, b = 181
Lösung 9: a = 64, b = 127
Lösung 10: a = 4, b = 229
Lösung 11: a = 8, b = 239
Lösung 12: a = 13, b = 256
Lösung 13: a = 64, b = 241
Lösung 14: a = 32, b = 311
Lösung 15: a = 8, b = 419
Lösung 16: a = 8, b = 449
Lösung 17: a = 71, b = 512
Lösung 18: a = 101, b = 512
Lösung 19: a = 32, b = 641
Lösung 20: a = 32, b = 821

A.6. N = 50.000

Lösung 1: a = 4, b = 13
Lösung 2: a = 4, b = 61
Lösung 3: a = 16, b = 73
Lösung 4: a = 16, b = 111
Lösung 5: a = 64, b = 73
Lösung 6: a = 32, b = 131
Lösung 7: a = 16, b = 163
Lösung 8: a = 4, b = 181
Lösung 9: a = 64, b = 127
Lösung 10: a = 4, b = 229
Lösung 11: a = 8, b = 239
Lösung 12: a = 13, b = 256
Lösung 13: a = 64, b = 241
Lösung 14: a = 32, b = 311
Lösung 15: a = 8, b = 419
Lösung 16: a = 8, b = 449
Lösung 17: a = 128, b = 419
Lösung 18: a = 71, b = 512
Lösung 19: a = 101, b = 512
Lösung 20: a = 32, b = 641
Lösung 21: a = 32, b = 701
Lösung 22: a = 32, b = 821
Lösung 23: a = 416, b = 827
Lösung 24: a = 512, b = 911

A.7. N = 100.000

Lösung 1: a = 4, b = 13
Lösung 2: a = 4, b = 61
Lösung 3: a = 16, b = 73
Lösung 4: a = 16, b = 111
Lösung 5: a = 64, b = 73
Lösung 6: a = 32, b = 131
Lösung 7: a = 16, b = 163
Lösung 8: a = 4, b = 181
Lösung 9: a = 64, b = 127
Lösung 10: a = 4, b = 229
Lösung 11: a = 8, b = 239
Lösung 12: a = 13, b = 256
Lösung 13: a = 64, b = 241
Lösung 14: a = 32, b = 311
Lösung 15: a = 8, b = 419
Lösung 16: a = 8, b = 449
Lösung 17: a = 128, b = 419
Lösung 18: a = 256, b = 313
Lösung 19: a = 71, b = 512
Lösung 20: a = 101, b = 512
Lösung 21: a = 128, b = 509
Lösung 22: a = 8, b = 659
Lösung 23: a = 32, b = 641
Lösung 24: a = 128, b = 569
Lösung 25: a = 32, b = 701
Lösung 26: a = 201, b = 556
Lösung 27: a = 128, b = 659
Lösung 28: a = 8, b = 809
Lösung 29: a = 32, b = 821
Lösung 30: a = 404, b = 503
Lösung 31: a = 512, b = 761
Lösung 32: a = 512, b = 911
Lösung 33: a = 32, b = 1601
Lösung 34: a = 512, b = 1151
Lösung 35: a = 512, b = 1181
Lösung 36: a = 32, b = 1721
Lösung 37: a = 32, b = 1811
Lösung 38: a = 5, b = 2048
Lösung 39: a = 224, b = 2039
Lösung 40: a = 512, b = 1781

A.8. N = 250.000

Lösung 1: a = 4, b = 13
Lösung 2: a = 4, b = 61
Lösung 3: a = 16, b = 73
Lösung 4: a = 16, b = 111
Lösung 5: a = 64, b = 73
Lösung 6: a = 32, b = 131
Lösung 7: a = 16, b = 163
Lösung 8: a = 4, b = 181

Lösung 9: a = 64, b = 127
Lösung 10: a = 4, b = 229
Lösung 11: a = 8, b = 239
Lösung 12: a = 13, b = 256
Lösung 13: a = 64, b = 241
Lösung 14: a = 32, b = 311
Lösung 15: a = 8, b = 419
Lösung 16: a = 8, b = 449
Lösung 17: a = 128, b = 419
Lösung 18: a = 256, b = 313
Lösung 19: a = 71, b = 512
Lösung 20: a = 101, b = 512
Lösung 21: a = 128, b = 509
Lösung 22: a = 8, b = 659
Lösung 23: a = 32, b = 641
Lösung 24: a = 128, b = 569
Lösung 25: a = 32, b = 701
Lösung 26: a = 201, b = 556
Lösung 27: a = 256, b = 523
Lösung 28: a = 128, b = 659
Lösung 29: a = 8, b = 809
Lösung 30: a = 64, b = 757
Lösung 31: a = 32, b = 821
Lösung 32: a = 256, b = 673
Lösung 33: a = 128, b = 839
Lösung 34: a = 40, b = 937
Lösung 35: a = 256, b = 733
Lösung 36: a = 421, b = 576
Lösung 37: a = 32, b = 1013
Lösung 38: a = 36, b = 1051
Lösung 39: a = 8, b = 1109
Lösung 40: a = 16, b = 1191
Lösung 41: a = 8, b = 1259
Lösung 42: a = 512, b = 761
Lösung 43: a = 8, b = 1289
Lösung 44: a = 8, b = 1319
Lösung 45: a = 128, b = 1217
Lösung 46: a = 128, b = 1229
Lösung 47: a = 8, b = 1373
Lösung 48: a = 512, b = 911
Lösung 49: a = 32, b = 1601
Lösung 50: a = 512, b = 1151
Lösung 51: a = 512, b = 1181
Lösung 52: a = 32, b = 1721
Lösung 53: a = 32, b = 1811
Lösung 54: a = 512, b = 1481
Lösung 55: a = 5, b = 2048
Lösung 56: a = 32, b = 2141
Lösung 57: a = 512, b = 1721
Lösung 58: a = 512, b = 1811
Lösung 59: a = 32, b = 2441
Lösung 60: a = 32, b = 2801
Lösung 61: a = 1424, b = 1559
Lösung 62: a = 512, b = 2591
Lösung 63: a = 512, b = 2801
Lösung 64: a = 1136, b = 2207
Lösung 65: a = 1616, b = 2027
Lösung 66: a = 1259, b = 3584

A.9. N = 500.000

Lösung 1: a = 4, b = 13
Lösung 2: a = 4, b = 61
Lösung 3: a = 16, b = 73
Lösung 4: a = 16, b = 111
Lösung 5: a = 64, b = 73
Lösung 6: a = 32, b = 131
Lösung 7: a = 16, b = 163
Lösung 8: a = 4, b = 181
Lösung 9: a = 64, b = 127
Lösung 10: a = 4, b = 229
Lösung 11: a = 8, b = 239
Lösung 12: a = 13, b = 256
Lösung 13: a = 64, b = 241
Lösung 14: a = 32, b = 311
Lösung 15: a = 8, b = 419
Lösung 16: a = 8, b = 449
Lösung 17: a = 128, b = 419
Lösung 18: a = 256, b = 313
Lösung 19: a = 71, b = 512
Lösung 20: a = 101, b = 512

Lösung 21: a = 128, b = 509
Lösung 22: a = 8, b = 659
Lösung 23: a = 32, b = 641
Lösung 24: a = 128, b = 569
Lösung 25: a = 32, b = 701
Lösung 26: a = 201, b = 556
Lösung 27: a = 256, b = 523
Lösung 28: a = 128, b = 659
Lösung 29: a = 8, b = 809
Lösung 30: a = 64, b = 757
Lösung 31: a = 32, b = 821
Lösung 32: a = 256, b = 673
Lösung 33: a = 128, b = 839
Lösung 34: a = 40, b = 937
Lösung 35: a = 256, b = 733
Lösung 36: a = 421, b = 576
Lösung 37: a = 32, b = 1013
Lösung 38: a = 36, b = 1051
Lösung 39: a = 8, b = 1109
Lösung 40: a = 16, b = 1191
Lösung 41: a = 8, b = 1259
Lösung 42: a = 512, b = 761
Lösung 43: a = 256, b = 1033
Lösung 44: a = 8, b = 1289
Lösung 45: a = 8, b = 1319
Lösung 46: a = 128, b = 1217
Lösung 47: a = 128, b = 1229
Lösung 48: a = 8, b = 1373
Lösung 49: a = 128, b = 1259
Lösung 50: a = 256, b = 1153
Lösung 51: a = 512, b = 911
Lösung 52: a = 8, b = 1499
Lösung 53: a = 128, b = 1409
Lösung 54: a = 32, b = 1601
Lösung 55: a = 16, b = 1641
Lösung 56: a = 512, b = 1151
Lösung 57: a = 128, b = 1553
Lösung 58: a = 128, b = 1559
Lösung 59: a = 512, b = 1181
Lösung 60: a = 8, b = 1709
Lösung 61: a = 128, b = 1619
Lösung 62: a = 32, b = 1721
Lösung 63: a = 3, b = 1756
Lösung 64: a = 512, b = 1289
Lösung 65: a = 768, b = 1039
Lösung 66: a = 32, b = 1811
Lösung 67: a = 8, b = 1949
Lösung 68: a = 8, b = 1979
Lösung 69: a = 512, b = 1481
Lösung 70: a = 5, b = 2048
Lösung 71: a = 32, b = 2141
Lösung 72: a = 512, b = 1721
Lösung 73: a = 512, b = 1811
Lösung 74: a = 32, b = 2441
Lösung 75: a = 32, b = 2741
Lösung 76: a = 32, b = 2801
Lösung 77: a = 512, b = 2411
Lösung 78: a = 512, b = 2441
Lösung 79: a = 32, b = 3041
Lösung 80: a = 512, b = 2591
Lösung 81: a = 512, b = 2621
Lösung 82: a = 512, b = 2801
Lösung 83: a = 32, b = 3371
Lösung 84: a = 512, b = 3041
Lösung 85: a = 32, b = 3581
Lösung 86: a = 32, b = 3911
Lösung 87: a = 512, b = 3461
Lösung 88: a = 32, b = 4001
Lösung 89: a = 1871, b = 2192
Lösung 90: a = 512, b = 3821
Lösung 91: a = 1259, b = 3584
Lösung 92: a = 2459, b = 2864
Lösung 93: a = 512, b = 5861
Lösung 94: a = 3299, b = 3584

Literaturverzeichnis

[Wikipedia 2008] Wikipedia: *Luzifer-Raetsel*. `http://de.wikipedia.org/wiki/Luzifer-Rätsel`, 2008