アルゴリズム特論
−数に関する話−

山本修身

四則演算について

- 四則演算はそれぞれ一定時間で計算できると仮定する．→ 本当は正しくない．
- なるべく少ない四則演算で計算することが望ましい．
- 桁の大きな整数を計算すると，非常に多くの時間を要する．

整数の最大公約数

- ユークリッドの互除法
- GCD(45, 30):
  45÷30 = 1...15
  30÷15 = 2...0 → GCD(45, 30) = 15

割り切れたら停止

greatest common divisor

Euclidの互除法のプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int gcd( int a, int b)
{
        if ( b == 0 ) return a;
        else {
                int c = a % b;
                return gcd( b, c);
        } /* if */
}
int main()
{
        int a = 60;
        int b = 45;
        int c = gcd( a, b );
        printf( "gcd(%d, %d) = %d\n", a, b, c );
        return 0;
}
```

Euclidアルゴリズムの実行結果

```
kauai:yama550> gcc euclid.c -o euclid
kauai:yama552> ./euclid
gcd(60, 45) = 15
kauai:yama553>
```

Euclidの互除法とは？

- Euclidは最大公約数を計算するためのアルゴリズムである.
- そもそも因数分解が一意的に行えない世界では意味がない．たとえば，a + b√-5，a, bは整数という世界では成り立たない.

$$6 = 2 \cdot 3 = (1 + \sqrt{-5})(1 - \sqrt{-5})$$

- 因数分解が一意的に行えるとしてもユークリッドの互除法が使えるとは限らない⇒使えるものを**ユークリッド環**と呼ぶ.

Euclidの互除法とは？（つづき）

- 整数や1変数多項式環はユークリッド環であることが知られている．だからユークリッドの互除法が使える！
- 多変数多項式は1通りに因数分解できるが，ユークリッドの互除法が使えない．**グレーブナー基底の理論 (Buchberger: 1980年代以降の研究)**

Euclidの互除法はなぜ正しいか（1）

- **止まるとすれば正しく計算することについて：**
  gcd(a, b) = gcd(b, a % b)である．なぜならば,
  a = xr, b = yr (rが最大公約数, xとyは互いに素)とすれば，a % b = xr % yr = (x % y) r であり，x % y と y は互いに素であるから gcd(a, b) = gcd(b, a%b)

```
int gcd( int a, int b)
{
        if ( b == 0 ) return a;
        else {
                int c = a % b;
                return gcd( b, c);
        } /* if */
}
```

b = 0ならばgcd(a, b)はaとなるから最大公約数が計算される

Euclidの互除法はなぜ正しいか（2)

- **このアルゴリズムが停止するということについて：**a とbの和考える．a, b > 0と仮定する．a + b > b + (a % b)であることは明らか．どんどん小さくなって行くので，いつか b = 0となる．b = 0ならば停止する．

```
int gcd( int a, int b)
{
        if ( b == 0 ) return a;
        else {
                int c = a % b;
                return gcd( b, c);
        } /* if */
}
```

Euclidの互除法の計算量

- 単純な整数についてのEuclidの互除法の計算量は$O(\log(n))$ である．但し，nは扱う数の最大値である［証明を考えてみよう］．
- 多項式についてのEuclidの互除法については非常に複雑である．

オーダとは何か？(1)

- 計算量は入力に依存して少なくも多くもなり得る．また，入力のサイズにも依存する．
- 普通，計算量の上界を求めて，それによってアルゴリズムの性能として記述する．
- 用いる計算機の性能によって，実際の計算時間は変化するので，定数倍は意味がない．そこで定数倍を無視した記述がオーダーの記述である．

オーダとは何か？(2)

- オーダの記述には3種類ある．

$$O(f(n)) = g(n) \Leftrightarrow g(n) \leq c \cdot f(n)$$

$$\Theta(f(n)) = g(n) \Leftrightarrow c_2 \cdot f(n) \leq g(n) \leq c_1 \cdot f(n)$$

$$\Omega(f(n)) = g(n) \Leftrightarrow c \cdot f(n) \leq g(n)$$

c, c1, c2は適当な定数としてあらかじめ指定できなければならない

オーダとは何か？(3)

- 練習問題：

$$g_1(n) = O(n^2), g_2(n) = O(n^3)$$

$$\Downarrow$$

$$g_1(n) + g_2(n) = O(n^3)$$

誰にでも納得できるように説明せよ.

多項式のGCD計算（1変数）

- 例：

$$\mathrm{GCD}(x^4 + x^2 + 1, x^3 - 1)$$

$$(x^4 + x^2 + 1) \div (x^3 - 1) = x \ldots x^2 + x + 1$$

$$(x^3 - 1) \div \underbrace{(x^2 + x + 1)}_{\mathrm{GCD}} = (x - 1) \ldots 0$$

多項式の次数は着実に落ちて行く．しかし，
係数が爆発してしまうことがある．

中間膨張形式の例

- 以下のGCDを計算してみよう！

$$\mathrm{CGD}(x^4 + 3x^3 + x + 1, 31x^2 + 1)$$

$$\frac{28}{31}x + \frac{962}{961}$$

$$\frac{237437}{6076} \Rightarrow 1$$

次数が高くなると，もっと悲惨な例がいくらでもつくれる

中間膨張形式の例(2)

- ちょっとだけ悲惨な例

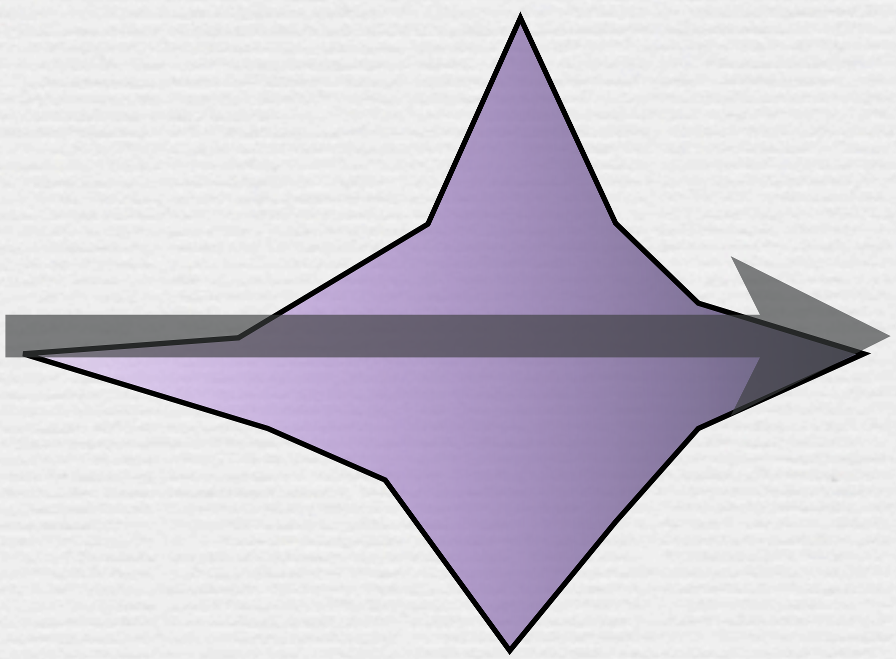
$$\mathrm{GCD}(x^{10}+3x^3+x+1,\,31x^2+1)$$

$$\Downarrow$$

$$\frac{28}{31}x+\frac{28629150}{28629151}$$

$$\Downarrow$$

$$\frac{210299529796381}{5392472365756} \Rightarrow 1$$



かけ算命令は必要か (1)

- かけ算は足し算を使って計算できるのではないか？足し算だけあれば十分か？
- しかし，下のようなプログラムでは遅すぎる．

```
int mult(int a, int b)
{
  int i;
  int s = 0;
  for (i = 0; i < b; i++ ){
    s = s + a;
  }
  return s;
} /* mult */
```

$$O(n + m)$$

かけ算命令は必要か (2)

- もう少し工夫すると，速くなる．

```
int mult2(int a, int b)
{
  if (b == 0) return 0;
  else if (b % 2 == 0){
    int mm = mult2(a, b / 2);
    return mm + mm;
  } else {
    int mm = mult2(a, b / 2);
    return mm + mm + a;
  } /* if */
} /* mult2 */
```

$$a \times 2b' = 2(a \times b')$$

$$a \times (2b' + 1) = 2(a \times b') + a$$

$$O(\log n + \log m)$$

ただし，足し算の計算時間は定数とする

それでは，足し算命令は必要か？

- 「２倍する」，「半分にする」と「１を足す」の3つの演算を用いれば作ることができる

$$O((\log n + \log m)^2)$$

```
int mult3(int a, int b)
{
  if (b == 0) return 0;
  else if (b % 2 == 0){
    int mm = mult3(a, b / 2);
    return mm * 2;
  } else {
    int mm = mult3(a, b / 2);
    return add2(mm * 2, a);
  } /* if */
} /* mult3 */
```

```
int add2(int a, int b)
{
  if (b == 0) return a;
  else if (a == 0) return b;
  else {
    int mm = add2(a / 2, b / 2);
    if (a % 2 > 0 && b % 2 > 0)
      return (mm + 1) * 2;
    else if (a % 2 > 0 && b % 2 == 0)
      return mm * 2 + 1;
    else if (a % 2 == 0 && b % 2 > 0)
      return mm * 2 + 1;
    else
      return mm * 2;
  }
} /* add2 */
```

数を累乗するのに
どのくらいの計算が必要か？

- 適当な整数を累乗することを考える．単純なかけ算は定数時間で計算できるとする．

$$x^n = x \cdot x \cdots x$$

- かけ算の回数は$O(n)$となる．これで良いのか？

数を累乗するのに
どのくらいの計算が必要か？

- もっと速く計算することができる．

```
int exp(int a, int n)
{
   if (n == 0) return 1;
   else {
      int m = exp(a, n / 2);
       m = m * m;
       if (n が奇数) m = m * a;

       return m;
}
```

$F(n)$を計算時間とすると，

$$F(n) = F(n/2) + c$$

$$F(n) = O(\log n)$$

累乗計算の例

- 2の1000乗は1000回かける必要はない.

$$
\begin{aligned}
a^{1000} &= (a^{500})^2 \\
a^{500} &= (a^{250})^2 \\
a^{250} &= (a^{125})^2 \\
a^{125} &= (a^{61})^2 a \\
a^{61} &= (a^{30})^2 a \\
a^{30} &= (a^{15})^2 \\
a^{15} &= (a^{7})^2 a \\
a^{7} &= (a^{3})^2 a \\
a^{3} &= a^2 a
\end{aligned}
$$

素数判定の話

- 与えられた数が素数であるかを判定する問題は数が大きくなるとそれほど簡単ではない.
- 数が小さいときには，端から割って行けば良い. 効率の良い割り方として，エラトステネスのふるいがある.
- 数が大きいと端から割って行くと止まらなくなる.

素数と２項係数のある性質

- pが素数であれば，任意の整数 $0 < i < p$ について

$$p \left| \begin{pmatrix} p \\ i \end{pmatrix} \right.$$

$a \mid b$　aはbを割るきる

$$\begin{pmatrix} p \\ i \end{pmatrix} = \frac{p!}{(p-i)!i!}$$

$$= \frac{p \times (p-1) \times \cdots \times 2 \times 1}{(p-i) \times (p-i-1) \times \cdots \times 2 \times 1 \times i \times (i-1) \times \cdots \times 2 \times 1}$$

pは素数であり，分母にpの倍数は存在しないので，この２項係数はpの倍数となる．

２項定理を思い出そう

- 2項定理とは$x + y$のベキ乗を展開する公式である.

$$(x+y)^p = \sum_{i=0}^{p} \left(\begin{array}{c} p \\ i \end{array} \right) x^i y^{p-i}$$

- pが素数のとき，$0 < i < p$ ならば，２項係数はpの約数である.

素数を法とする場合

- 素数pを法とすると，きれいな性質が得られる．

$$
\begin{aligned}
(x+y)^p &= \sum_{i=0}^{p} \left(\begin{array}{c} p \\ i \end{array} \right) x^i y^{p-i} (\mathrm{mod} p) \\
&= x^0 y^p + x^p y^0 \pmod{p} \\
&= x^p + y^p \pmod{p}
\end{aligned}
$$

フェルマーの小定理

- 実はこの性質を用いるともっと強い性質が得られる．

$$a^p = (\underbrace{1+1+\cdots+1}_{a個})^p$$

$$= \underbrace{1^p+1^p+\cdots+1^p}_{a個} (\mathrm{mod}\ p)$$

$$= \underbrace{1+1+\cdots+1}_{a個} (\mathrm{mod}\ p)$$

$$= a (\mathrm{mod}\ p)$$

$$\underline{a^{p-1} = 1 (\mathrm{mod}\ p)}$$

フェルマーの小定理の例

N = 13について		N = 15について	
0	0	0	0
1	1	1	1
2	1	2	4
3	1	3	9
4	1	4	1
5	1	5	10
6	1	6	6
7	1	7	4
8	1	8	4
9	1	9	6
10	1	10	10
11	1	11	1
12	1	12	9
		13	4
		14	1

```c
#include <stdio.h>
#include <stdlib.h>

int myexp(int x, int i, int p);

int myexp(int x, int i, int p) {
  if ( i == 0 ) return 1;
  else {
    int j =  myexp( x, i / 2, p );
    j = (j * j) % p;
    if (i % 2 == 1)
      j = (j * x) % p;
    return j;
  } /* if */
} /* myexp */

int main() {
  enum{ N = 13};
  int i;
  for (i = 0; i < N; i++ ){
    printf( "%5d%5d\n", i, myexp( i, N - 1, N ));
  } /* for */
  return 0;
} /* main */
```

$$a^{p-1} = 1 (\mathrm{mod}\ p)$$

フェルマーテスト

- 不完全なテストであるが大抵うまく動くテスト

(1) 2以上n未満の整数を適当に選択する．これをaとおく

(2) GCD(a, n)が1でなければ，その因子がnの約数となりnは素数ではない．

(3) $a^{n-1} \mathrm{mod}\ n$ を計算し，1でなければ，素数ではない．(1), (2), (3)を繰り返す

フェルマーテストをくぐり抜ける数

- フェルマーテストによって素数でないことが証明できない数のことを擬素数と呼ぶ．すなわち，n と互いに素な数aに対して，

$$a^{n-1} = 1\cancel{(\mathrm{mod}\ p)} \qquad (\mathrm{mod\ n})$$

常に成り立つ数が存在する．このような数をカーマイケル数と呼ぶ．カーマイケル数は限りなく存在することが知られている．

カーマイケル数（擬素数）の例

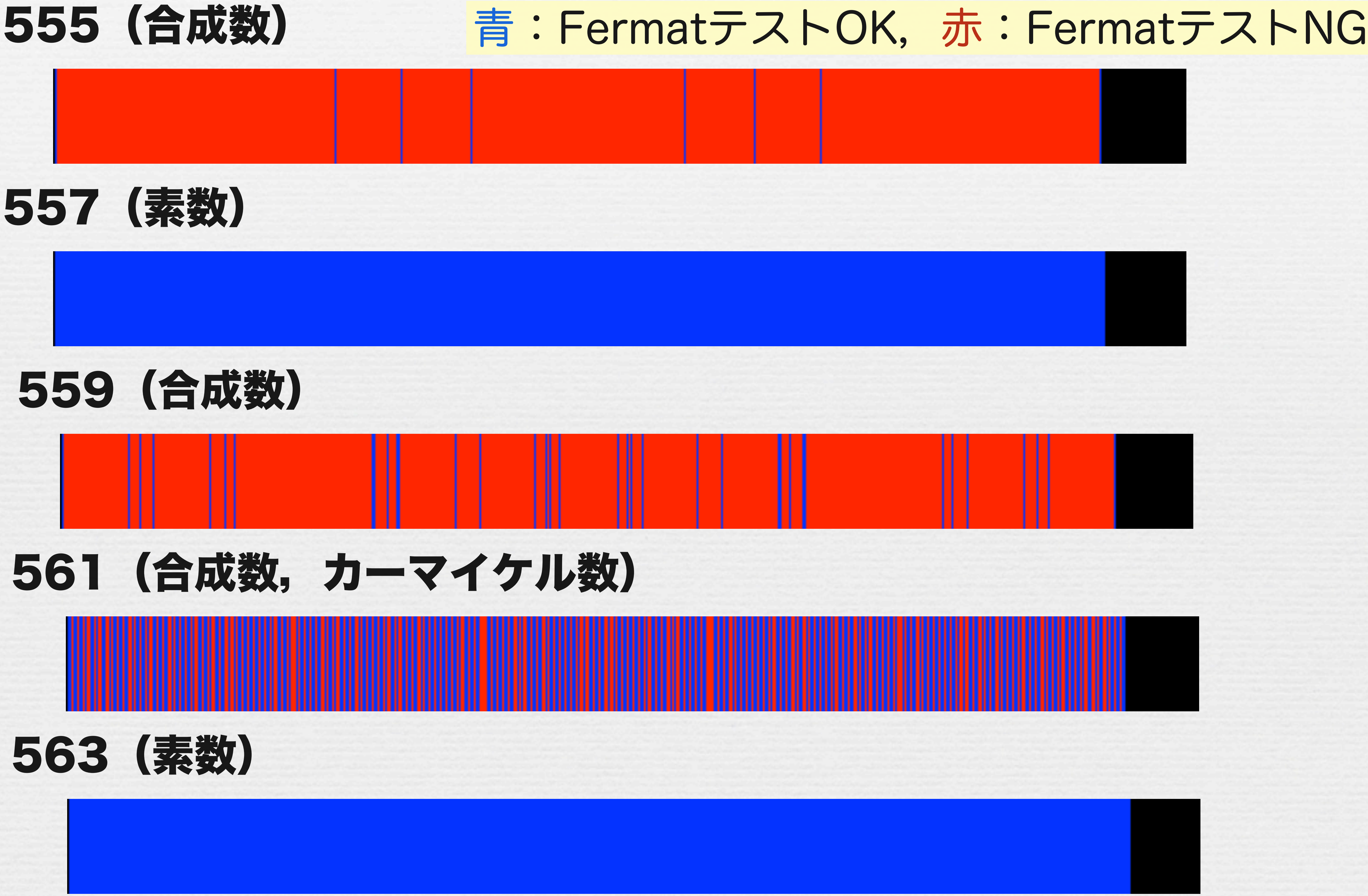
- $561 = 3 \times 11 \times 17$ はカーマイケル数である.

0	0	43	1	86	1	129	375	172	1
1	1	44	154	87	375	130	1	173	1
2	1	45	375	88	154	131	1	174	375
3	375	46	1	89	1	132	528	175	1
4	1	47	1	90	375	133	1	176	154
5	1	48	375	91	1	134	1	177	375
6	375	49	1	92	1	135	375	178	1
7	1	50	1	93	375	136	34	179	1
8	1	51	408	94	1	137	1	180	375
9	375	52	1	95	1	138	375	181	1
10	1	53	1	96	375	139	1	182	1
11	154	54	375	97	1	140	1	183	375
12	375	55	154	98	1	141	375	184	1
13	1	56	1	99	528	142	1	185	1
14	1	57	375	100	1	143	154	186	375
15	375	58	1	101	1	144	375	187	187
16	1	59	1	102	408	145	1	188	1
17	34	60	375	103	1	146	1	189	375
18	375	61	1	104	1	147	375	190	1
19	1	62	1	105	375	148	1	191	1
20	1	63	375	106	1	149	1	192	375
21	375	64	1	107	1	150	375	193	1
22	154	65	1	108	375	151	1	194	1
23	1	66	528	109	1	152	1	195	375
24	375	67	1	110	154	153	408	196	1
25	1	68	34	111	375	154	154	197	1
26	1	69	375	112	1	155	1	198	528
27	375	70	1	113	1	156	375	199	1
28	1	71	1	114	375	157	1	200	1
29	1	72	375	115	1	158	1	201	375
30	375	73	1	116	1	159	375	202	1
31	1	74	1	117	375	160	1	203	1
32	1	75	375	118	1	161	1	204	408
33	528	76	1	119	34	162	375	205	1
34	34	77	154	120	375	163	1	206	1
35	1	78	375	121	154	164	1	207	375
36	375	79	1	122	1	165	528	208	1
37	1	80	1	123	375	166	1	209	154
38	1	81	375	124	1	167	1	210	375
39	375	82	1	125	1	168	375	211	1
40	1	83	1	126	375	169	1	212	1
41	1	84	375	127	1	170	34	213	375
42	375	85	34	128	1	171	375	214	1

215	1	259	1	303	375	347	1	391	34	435	375	479	1
216	375	260	1	304	1	348	375	392	1	436	1	480	375
217	1	261	375	305	1	349	1	393	375	437	1	481	1
218	1	262	1	306	408	350	1	394	1	438	375	482	1
219	375	263	1	307	1	351	375	395	1	439	1	483	375
220	154	264	528	308	154	352	154	396	528	440	154	484	154
221	34	265	1	309	375	353	1	397	1	441	375	485	1
222	375	266	1	310	1	354	375	398	1	442	34	486	375
223	1	267	375	311	1	355	1	399	375	443	1	487	1
224	1	268	1	312	375	356	1	400	1	444	375	488	1
225	375	269	1	313	1	357	408	401	1	445	1	489	375
226	1	270	375	314	1	358	1	402	375	446	1	490	1
227	1	271	1	315	375	359	1	403	1	447	375	491	1
228	375	272	34	316	1	360	375	404	1	448	1	492	375
229	1	273	375	317	1	361	1	405	375	449	1	493	34
230	1	274	1	318	375	362	1	406	1	450	375	494	1
231	528	275	154	319	154	363	528	407	154	451	154	495	528
232	1	276	375	320	1	364	1	408	408	452	1	496	1
233	1	277	1	321	375	365	1	409	1	453	375	497	1
234	375	278	1	322	1	366	375	410	1	454	1	498	375
235	1	279	375	323	34	367	1	411	375	455	1	499	1
236	1	280	1	324	375	368	1	412	1	456	375	500	1
237	375	281	1	325	1	369	375	413	1	457	1	501	375
238	34	282	375	326	1	370	1	414	375	458	1	502	1
239	1	283	1	327	375	371	1	415	1	459	408	503	1
240	375	284	1	328	1	372	375	416	1	460	1	504	375
241	1	285	375	329	1	373	1	417	375	461	1	505	1
242	154	286	154	330	528	374	187	418	154	462	528	506	154
243	375	287	1	331	1	375	375	419	1	463	1	507	375
244	1	288	375	332	1	376	1	420	375	464	1	508	1
245	1	289	34	333	375	377	1	421	1	465	375	509	1
246	375	290	1	334	1	378	375	422	1	466	1	510	408
247	1	291	375	335	1	379	1	423	375	467	1	511	1
248	1	292	1	336	375	380	1	424	1	468	375	512	1
249	375	293	1	337	1	381	375	425	34	469	1	513	375
250	1	294	375	338	1	382	1	426	375	470	1	514	1
251	1	295	1	339	375	383	1	427	1	471	375	515	1
252	375	296	1	340	34	384	375	428	1	472	1	516	375
253	154	297	528	341	154	385	154	429	528	473	154	517	154
254	1	298	1	342	375	386	1	430	1	474	375	518	1
255	408	299	1	343	1	387	375	431	1	475	1	519	375
256	1	300	375	344	1	388	1	432	375	476	34	520	1
257	1	301	1	345	375	389	1	433	1	477	375	521	1
258	375	302	1	346	1	390	375	434	1	478	1	522	375

いくつかの数に対するFermatテスト

- 素数か否かによってかなり状況は異なる



フェルマーテストの高度化・改良

- Miller-Rabinのアルゴリズム（1980)
- 擬素数の存在により，フェルマーテストには欠陥があり，いくらやっても素数でない数を素数と判定する可能性がある．
- Miller-Rabinのアルゴリズムも確率的であるが，判定の際の誤り確率を見積もることができ，それを繰り返しによって確実に減らすことが可能である．

mod pの世界についての準備

- mod pの世界で以下の単純な2次方程式を考える．

$$x^2 = 1 (\mathrm{mod} \ \ p)$$

$$(x-1)(x+1) = 0 (\mathrm{mod} \ \ p)$$

pが素数であることから，$(x - 1)$または$(x + 1)$はpで割り切れる．従って，

$$x - 1 = 0 (\mathrm{mod} \ \ p) \quad \text{または} \quad x + 1 = 0 (\mathrm{mod} \ \ p)$$

$$x = \pm 1 (\mathrm{mod} \ \ p) \quad \text{これしか解はない！}$$

素数について成り立つこと (1)

- pが素数であると仮定するとつぎの命題が成り立つ
  - p - 1が2^s・dと表現できるとする.

$$a^d = 1(\mathrm{mod} n)$$

または,

$$0 \leq \exists i < s, a^{2^i d} = -1(\mathrm{mod} n)$$

フェルマーの小定理より

素数について成り立つこと (2)

- これより,

$$a^d \neq 1(\text{mod n})$$

かつ

$$0 \leq \forall i < s, a^{2^i d} \neq -1(\text{mod} n)$$

ならば，pは素数ではない．

- このことから，素数でない場合には，100%わかる．問題は素数のときに素数であると言えるかということ．

Miller-Rabinのアルゴリズム

- nが与えられたら，n - 1 = 2^{s}・dに分解する．
- [1, n-1]の範囲からランダムにaを選ぶ．
- $a^d \neq 1(\text{mod n})$ かつ $0 \leq \forall i < s, a^{2^i d} \neq -1(\text{mod} n)$ であることが確認されれば，合成数である．そうでなければ，「多分素数である」と出力する．

Miller-Rabinの定理

- 合成数 p が与えられているのに，p について「多分素数である」と返す確率は1/4以下である．
- 従って，このアルゴリズムを20回以上行えば，合成数なのに，素数であると返す確率は$(1/4)^{20}$であり，これは，$1/10^{12}$よりも小さい

いくつかの数に対するMRテスト

- 素数か否かによってかなり状況は異なる

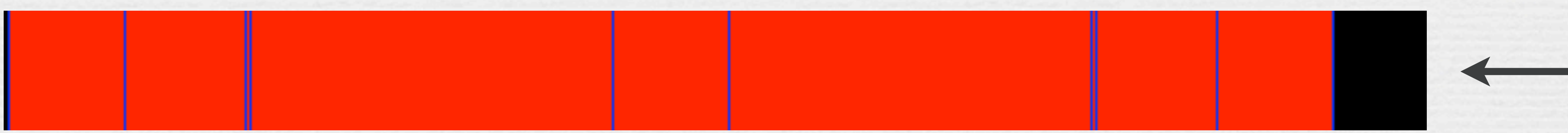
555（合成数）

青：MRテストOK，赤：MRテストNG

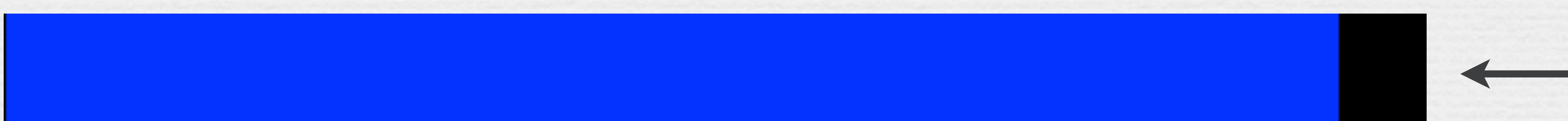
557（素数）

559（合成数）

561（合成数，カーマイケル数）



563（素数）



MRアルゴリズムをベースにした 決定的（確率的でない）アルゴリズム

- 拡張リーマン予想が正しいということを認めれば，2から2 $(\ln n)^2$ までの整数についてMRアルゴリズムによって，素数でないということが言えなければ，素数であることが知られている．
- ただし，これはあくまで，拡張リーマン予想が正しければ正しいということである．
- 前述の563の場合，2から12までの整数についてMRテストを行ってパスすれば素数である．

AKS素数判定法

- 2002年にインド工科大学のAgrawalらによって発見された非確率的アルゴリズム．素数判定は$O((\log n)^{7.5})$で計算可能であることが示された．オーダは大きいが，リーマン予想を仮定しない非確率的アルゴリズム（決定的アルゴリズム）としては大発見であった．

PRIMES is in P

RSA暗号系

- 1977年にMITの3人によって発明された暗号システム．デジタル署名の技術を含む．
- この技術をつかって，PGP (Pretty Good Privacy)というシステムが実際に作られた．
- Fermatの小定理を知っていれば，原理は高校生でも理解できる．

R. L. Rivest, A. Shamir, and L. Adelman: *A method for obtaining digital signature and public-key cryptsystems*. MIT Laboratory for Computer Science Technical Memo LCS/TM82; April 4,1977 (Revised December 12,1977).

RSAの原理(1)

- 2つの大きな桁の素数p, qを用意する．このとき，$n = pq$, $n' = (p - 1)(q - 1)$とおく．

- n'と互いに素な数 e をランダムに選択する．

- さらに$de \equiv 1 \pmod{n'}$となるdを計算する．

Message	→	暗号文	→	復号文
m		m^e		$(m^e)^d = m^{de}$

すべてmod nで計算

RSAの原理(2)

$$de = 1(\mathrm{mod\ n'})$$

$$c = m^e(\mathrm{mod\ n})$$ より，

$$c^d = (m^e)^d = m^{ed}(\mathrm{mod\ n})$$

$$m^{n'} = (m^{q-1})^{p-1} = 1(\mathrm{mod}\ p)$$

$$m^{n'} = (m^{p-1})^{q-1} = 1(\mathrm{mod}\ q)$$

中国式剰余定理より

$$m^{n'} - 1 = 0(\mathrm{mod}\ pq)$$

$de = n'k + 1$ より

$$m^{de} = m^{n'k+1} = (m^{n'})^k \cdot m = m\ (\mathrm{mod}\ pq)$$

RSAの原理(3)

- $n = pq$の因数分解ができない場合，nとeが与えられただけでは，dを計算することが困難である．同様に，nとdが与えられた状況でeを計算することも困難である．
- これがRSA暗号系が解けない理由である．
- eを公開鍵とよび，dを秘密鍵と呼ぶ．

デジタル署名

- 秘密鍵を使ってある適当なメッセージを暗号化し，元のメッセージと共に相手に送信することによって，このメッセージが秘密鍵の持ち主によるものであることを証明できる．
- 受け取った人は，秘密鍵で暗号化された情報を公開鍵で復号化してメッセージと比較してみれば良い．

プログラム例 (1)

```
public static BigInteger find_a_prime_number(BigInteger start,
                                              Random r){
    BigInteger ii = start.subtract( one );
    while ( true ){
        ii = ii.add( one );
        if (ii.remainder( two ).equals( zero ) ) continue;
        if (ii.remainder( three ).equals( zero ) ) continue;
        if (ii.remainder( five ).equals( zero ) ) continue;
        if (miller( ii, r )) return (ii );
    } /* while */
}
```

プログラム例 (2)

```
public static void main2(String [] args){
    init();
    Random r = new Random(22345);
    BigInteger a = new BigInteger( 300, r );
    BigInteger b = new BigInteger( 300, r );
    a = find_a_prime_number( a , r );
    b = find_a_prime_number( b , r );
    System.out.println( "a = " + a  );
    System.out.println( "b = " + b  );
    BigInteger n = a.multiply( b );
    BigInteger n_prime = (a.subtract(one)).multiply( b.subtract(one) );
    System.out.println( "n = " + n );
    System.out.println( "n' = " + n_prime );
    BigInteger d = find_a_prime_number(new BigInteger( 500, r ), r);
    BigInteger e = d.modInverse( n_prime );
    System.out.println( "d = "+ d );
    System.out.println( "e = "+ e );
    BigInteger c = new BigInteger( "123456789" );
    BigInteger encoded = expmod( c, d, n );
    BigInteger decoded = expmod( encoded, e, n);
    System.out.println( "c = " + c );
    System.out.println( "encoded =" + encoded);
    System.out.println( "decoded =" + decoded);
}
```

実行結果

```
a = 166668074238319095483313546042990947432076135127096026513645272376316331039
0684907534046963

b = 133305618380379397594032562225703265024920539136097753572566920240751206711
0208280674803709

n = 222177907006061079815334359575730124160828951112884349135332485574849003919
3214942879356598738082496922436614953035741355935697376042329330477481581435427
872841292885750574412585767

n' = 22217790700606107981533435957573012416082895111288434913533248557484900391
9321494287935659573834557073545168417957465866899357280607558669853968071931350
1702165915384857386203735096

d = 159040641995398294421005061364774427976103908828264615351996517961093247892
3382107137518716560033421324965621956837468331087014569786875435923450131581

e = 106752029135632100796995566769927316289821765733062164417281902269045941901
6782484543366633884189297779952793945667074200027960781289367191066456394493439
150150543111690733148025677

c = 123456789

encoded =1601848530662595872999393723670476854582212286579267673708220429161893
0941364830180121093431946431822904101539789696936671900827330147406873266524813
04260648899934855151235789789423

decoded =123456789
```

レポート問題 2

1. $x = 1\times10^{80}$から$x + 5\times10^{10}$の間にある素数の個数を求めよ．プログラムや考え方，計算に要した時間など詳しく述べること
2. 500,000より小さなカーマイケル数をすべて求め，これらの数についてFermatテストとMRテストを試した結果を示せ.

- 計算プログラムは授業で示したものを用いてよい．自分で変形して利用せよ．プログラムや実行結果をレポートに添付すること．また，計算の内容などを何も知らない人が読んでもわかるように解説すること.
- 提出期限は2週間後（6月11日）の授業まで.