データ構造とアルゴリズム実験レポート
課題１：<Java によるプログラミングの復讐>

<201811320><1 クラス><江頭輝>

締切日：<2019 年 4 月 22 日>
提出日：<2019 年 4 月 21 日>

1　必須課題

この課題では，教科書リスト 1-1 (p.3) を基にした最大公約数を求める Java プログラムの GCDIter.java を，教科書リスト 1-4 (p.7) の「ユークリッドの互除法」に基づいた Java プログラム GCDEuclid.java を作成した，また作成したプログラムのリストおよび実行結果を示した．

1.1　GCDEuclid.java の作成

1.1.1　実装の方針

まず，`GCDEuclid` クラスを定義し，２つの引数を取ってユークリッドの互除法を用いて算出される最大公約数を返す `euclid` メソッドとして，`euclid` メゾッドを実行しその結果を表示するのを `main` メソッドとして，クラス内にそれぞれ実装した．また，`main` メソッドは，最大公約数を求める元となる二つの引数 `n` と `m` をコマンドライン引数で渡すことによって動作する．このため，配列の長さと探索データに応じた探索時間の変化を調べるための実験が容易にできるようになった．また、assert 文による確認コードを実装し事前にデバッグしやすくしました。

1.1.2　実装コードおよびコードの説明

図 1GCDEuclid.java のソースコードを示す．1.1.1 節で述べた，`euclid` メソッドと `main` メソッドは，それぞれ 20~34 行目，2~17 行目の部分に相当する．

`euclid` メソッドは，`n` と `m` の最大公約数を求める。割ったあまりがゼロになるまで繰り返し割り算を行うのがユークリッドの互除法だが、そのまえに割られる数が割る数より大きいことが前提条件としてある。なので、23~27 行目で n が m より小さくなるように入れ替えを行っている。28~32 行目で実際にユークリッドの互除法を行った。n を m で割った余りを r として n, m の値を更新している。そして、余り（ここでは m）がゼロになった時の割る数（ここでは n）の値が最大公約数となる。

`main` メソッドは，`euclid` メソッドのテストケースとして機能する．実際に assert 分では euclid メゾッドを呼び出して、13673 と 31640 の最大公約数が 113 になるかを確認する。求める対象である `n` および `m` は，コマンドライン引数によってセットされ，ソースコードの 10，11 行目がそれに相当する．`main` メソッドに渡せるコマンドライン引数の数は必ず 2 個であるようにするため，5~8 行目でコマンドライン引数の配列数が

```
1  public class GCDEuclid {
2      public static void main(final String[] args) {
3          assert(euclid(13673, 31640) == 113);
4
5          if (args.length != 2) {
6              System.err.println("Usage:␣java␣GCDEuclid␣<int1>␣<int2>");
7              System.exit(1);
8          }
9          // Process arguments.
10         int n = Integer.parseInt(args[0]);
11         int m = Integer.parseInt(args[1]);
12         // Find the greatest common divisor.
13         int gcd = euclid(n, m);
14         System.out.println("The␣GCD␣of␣" + m + "␣and␣" + n + "␣is␣" + gcd + ".") ;
15     }
16
17     // Find the greatest common divisor of the two integers, n and m.
18     static int euclid(int n, int m) {
19         int r = 0;
20         // Let n be the smaller number.
21         if (n > m) {
22             int tmp = m;
23             m = n;
24             n = tmp;
25         }
26         //      n % m = x ... r
27         // ==> m % r = x' ... r'
28         while (m != 0) {
29             r = n % m;
30             n = m;
31             m = r;
32         }
33         return n;
34     }
35 }
```

図 1　GCDEuclid.java のソースコード

二つでない場合は．”Usage: java GCDEuclid ¡int1¿ ¡int2¿”というエラー文が帰ってきて実行が終了するように書かれている。そのチェックの後、n, m にコマンドライン引数をわたし、13 行目で euclid メゾッドを呼び出して変数 gcd に最大公約数を渡す。最後に”The GCD of ”+ m +” and ”+ n +” is ”+ gcd +”.”の形で最大公約数が出力される。

1.1.3　実行結果

まず，GCDEuclid.java を以下のコマンドでコンパイルする．

```
--------------------------------------
$ javac -g -verbose GCDEuclid.java
--------------------------------------
```

ここで，-g はデバッグ情報の付加させるためのオプション，-verbose はコンパイルの詳細な出力を表示させるためのオプションである．コンパイルが成功すると GCDEuclid.class が生成され，その後以下のコマンドを入力することによって，プログラムを実行できる．

```
--------------------------------------
$ java -ea GCDEuclid 57 76
The GCD of 76 and 57 is 19.
```

```
------------------------------------
```

57 と 76 の最大公約数を求める。すると、19 と返ってきた。$57 = 19 \times 3$, $76 = 19 \times 4$ であり確かに 57 と 76 の最大公約数は 19 なのでこのコードの挙動が正しいことが分かる。`-ea` オプションは assert 式を有効にするためのオプションであり、ここでもエラーは起こらなかった。

1.1.4 考察

最大公約数は、引数が自然数であることを前提とするため、自然数での確認のみにとどめておく。実際はバグを減らすために引数を自然数以外を受け付けないようにしなければならない。またこの時の実行時間はラメの定理より `O(log(n))` であることがわかる、これはよいアルゴリズムであるということなので最大公約数を求めるときは GCDIter.java よりも推奨される書き方だということが分かる。30, 31 行目の処理を減らすことでさらなる改善も期待できそうだ。

2 発展課題

2.1 時間計算量

この課題では，必須課題で作成した GCDIter.java，GCDEuclid.java について，それぞれの時間計算量を議論した．

一般的に，時間計算量はループにおける繰り返し回数によって左右されると言っても良い．GCDIter.java では，ループの繰り返し回数は，最小で 1 回，最大で n 回となり，平均は約 $n/2$ 回である．一回のループで 2 度の計算が行われ、n がパラメータであるので，計算量は $O(n)$ となる．すなわち，n を大きくするにつれて，探索時間は線形に増加する．

一方，BinarySearch.java では，ラメの定理より n の桁数の訳 5 倍桁数は $logn$ で表すので，平均は約 $logn$ 回である．これも n がパラメータであることに変わりはないので，計算量は $O(logn)$ となる．このことからも計算量が格段に減ることが分かる。

2.2 GCDRecursive.java の作成

2.2.1 実装の方針

実装の方針は，GCDEuclid.java と同様である．余り（m）がゼロになった時を終了条件とする再帰メゾッドを実装する。前回と同様に n が m より小さくなるような入れ替えをおこなったあと、n を m で割った余りを r とおく。recursive メゾッドを返すことでmがゼロになるまでループし続ける。

2.2.2 実装コードおよびコードの説明

図 2GCDRecursive.java のソースコードを示す．1.2.1 節で述べた，`recursive` メソッドと `main` メソッドは，それぞれ 19~31 行目，2~16 行の部分に相当する．今回はデバッグのスピードを速めるために、コマンドライン引数がない場合は n に 13673 を m に 31640 を代入させるようにした。本番環境ではコメントアウトすることが望ましい。`recursive` メゾッドは再帰メゾッドなのではじめに終了条件の m がゼロの時を書き、そこで n を最大公約数として返すようにしている。24~28 行目で入れ替えを行った後、r に n と m のあまりを代入して、m と r で再帰させている。

```java
public class GCDRecursive {
    public static void main(final String[] args) {
        // Process arguments.
        int n = 13673;
        int m = 31640;
        if (args.length != 2) {
            System.err.println("Usage: java GCDEuclid <int1> <int2>");
            // System.exit(1);
        } else {
            n = Integer.parseInt(args[0]);
            m = Integer.parseInt(args[1]);
        }
        int gcd = recursive(n, m);
        // gcd = 113
        System.out.println("The GCD of " + m + " and " + n + " is " + gcd + ".") ;
    }

    // Find the greatest common divisor of the two integers, n and m.
    static int recursive(int n, int m) {
        if (m == 0) {
            return n;
        }
        // Let n be the smaller number.
        if (n < m) {
            int tmp = m;
            m = n;
            n = tmp;
        }
        int r = n % m;
        return recursive(m, r);
    }
}
```

図 2　GCDRecursive.java のソースコード

2.2.3　実行結果

1.1.3 節で述べたコンパイル方法と同様に，GCDRecursive.java を `javac` コマンドでコンパイルし，GCDRecursive.class を生成する．その後，以下のコマンドを入力することによって，GCDRecursive.java のプログラムを実行する．

```
-------------------------------------
$ java -ea GCDRecursive 57 76
The GCD of 76 and 57 is 19.
-------------------------------------
```

2.2.4　考察

30 行目の再帰させるとき n のところに m, m のところに r を引数とすることで、前回の GCDEuclid.java で行った値の最代入を省略している。それにより、計算量は同じ `O(log(n)` だが多少の改善が行われたと思う。