データ構造とアルゴリズム実験レポート
課題２：<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 27 日>

1　必須課題

この課題では，セルに整数値を格納する連結リストを実現するクラス List を Java 言語で実装する。

1.1　List.java の作成

1.1.1　実装の方針

まず, List クラスを定義する。セルの直後（ポインタは）‘p.next‘のような形で表現し、セルの値は ‘p.data‘のように表記する。

1.1.2　実装コードおよびコードの説明

$insert_cell$ では $insert_cell$ では $delete_cell$ では $delete_cell_top$ では $display$ では

1.1.3　実行結果

まず，List.java を以下のコマンドでコンパイルする．

```
-------------------------------------
$ javac -g -verbose List.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 List.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 List 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

```java
public class List {
    static List head;
    List next;
    int data;

    static void insert_cell(List p, int d) {
        List new_cell = new List();
        new_cell.data = d;
        new_cell.next = p.next;
        p.next = new_cell;
    }

    static void insert_cell_top(int d) {
        List new_cell = new List();
        new_cell.data = d;
        new_cell.next = head;
        head = new_cell;
    }

    static void delete_cell(List p) {
        List q = p.next;
        p.next = q.next;
    }

    static void delete_cell_top() {
        List q = head;
        head = q.next;
    }

    static void display() {
        List tmp = head;
        while (tmp != null) {
            System.out.print(tmp.data);
            tmp = tmp.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        insert_cell_top(1);
        insert_cell(head, 3);
        insert_cell(head, 2);
        display();

        delete_cell(head);
        display();

        delete_cell_top();
        display();
    }
}
```

図 1　List.java のソースコード

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，List.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　QueueArray.java の作成

2.2.1　実装の方針

実装の方針は，List.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**??**QueueArray.java のソースコードを示す．**??**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。

2.2.3　実行結果

??節で述べたコンパイル方法と同様に，QueueArray.java を `javac` コマンドでコンパイルし，QueueArray.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，QueueArray.java のプログラムを実行する．

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

```java
import java.lang.reflect.Array;

public class QueueArray {
    int length, front, rear;
    int[] queue;

    // 指定された長さの配列を生成するコンストラクタ
    QueueArray(int len) {
        queue = new int[len];
        length = len;
        front = 0;
        rear = 0;
    }

    // データのエンキュー
    void enqueue(int val) {
        if (rear > length-1) {
            System.err.println("Queue␣Overflow␣!!");
            System.exit(1);
        }
        queue[rear] = val;
        rear += 1;
    }

    // データのデキュー
    int dequeue() {
        if (front == rear) {
            System.err.println("Queue␣Underflow␣!!");
            System.exit(1);
        }
        int x = queue[front];
        front += 1;
        return x;
    }

    // キューの要素の表示
    void display() {
        for (int i = front; i < rear; i++) {
            System.out.print(queue[i]);
        }
        System.out.println();
    }

    // main メソッド
    public static void main(String[] args) {
          QueueArray queue = new QueueArray(10);

          queue.enqueue(1);
          queue.enqueue(2);
          queue.display();

          System.out.println(queue.dequeue());
          System.out.println(queue.dequeue());
    }
}
```

図 2　QueueArray.java のソースコード

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 List.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。