山东大学

信息科学与工程学院

2025—2026 学年第一学期

实 验 报 告

课程名称： Java 编程技术

实验名称： 循环递归与算法优化

专业班级 通信一班

学生学号 202300120317

学生姓名 陈都阳

实验时间 2025年9月16日

【实验目的】

1. 熟悉Java的循环递归。
2. 尝试使用Java来优化复杂度。

【实验要求】

1. 按要求完成实验一到实验八。
2. 编译运行，并截图实验结果。
3. 实验后回答相关思考问题。

【第一个实验具体内容】

代码框的问题，代码部分的注释翻译成英文了，原版代码请看附件。

流程图

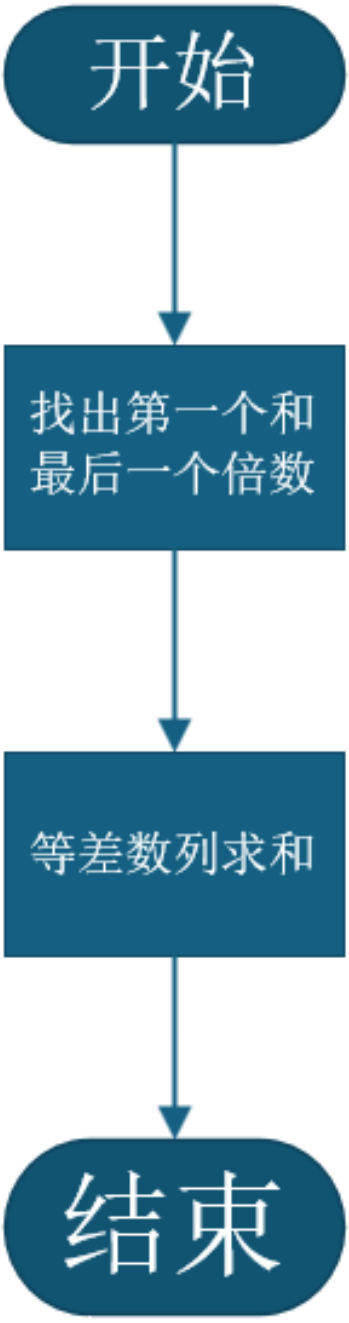

图 1: 流程图

部分源代码

由于代码较长，只显示主要部分，完整代码见附录。

```java
// One.java
public void calculateByFormula() {
    // Find the first and last multiples within the range
    int firstMultiple = (start % divisor == 0) ? start :
       start + (divisor - start % divisor);
    int lastMultiple = end - (end % divisor);
    // If there are no multiples in the range
    if (firstMultiple > end) {
        sum = 0;
        return;
    }
    int count = (lastMultiple - firstMultiple) / divisor + 1;
    sum = count * (firstMultiple + lastMultiple) / 2;
}
public static void main(String[] args) {
    One calculator = new One(1, 2000, 3);
    calculator.calculateByFormula();
    calculator.printResult("Using arithmetic progression
       formula");
}
```

图 2: One 主要部分源代码

实验过程与结果

```
PS F:\java_code\git>  & 'E:\java8_s\bin\jav
\User\workspaceStorage\60c6439be9e3dc2adb2c
 'ThreeJavaExam.One'
通过等差数列求和: :
1-2000之间所有3的倍数之和为: 666333
PS F:\java_code\git>
```

图 3: 运行结果

【第二个实验具体内容】

流程图

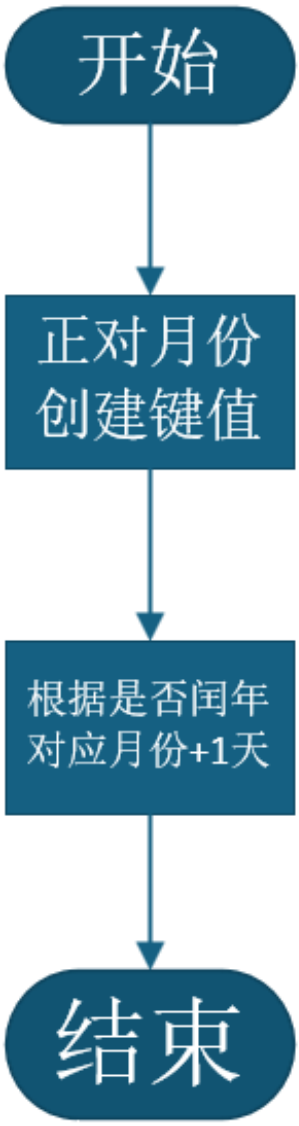

图 4: 流程图

部分源代码

```java
// Two.java
public int getDaysByArray() {
    if (month < 1 || month > 12) {
        return -1;
    }

    // Array of days in each month (index 0 unused for easy mapping)
    int[] daysInMonth = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    // If it's February in a leap year, return 29 days
    if (month == 2 && isLeapYear()) {
        return 29;
    }
    return daysInMonth[month];
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.println("===== Experiment 2: Date Calculator =====");
    System.out.print("Enter year: ");
    int year = scanner.nextInt();

    System.out.print("Enter month (1-12): ");
    int month = scanner.nextInt();
    // Create the date calculator object
    Two calculator = new Two(year, month);

    // Display detailed information
    calculator.printDetails();
    scanner.close();
}
```

图 5: TwoQuestion 部分源代码

实验过程与结果

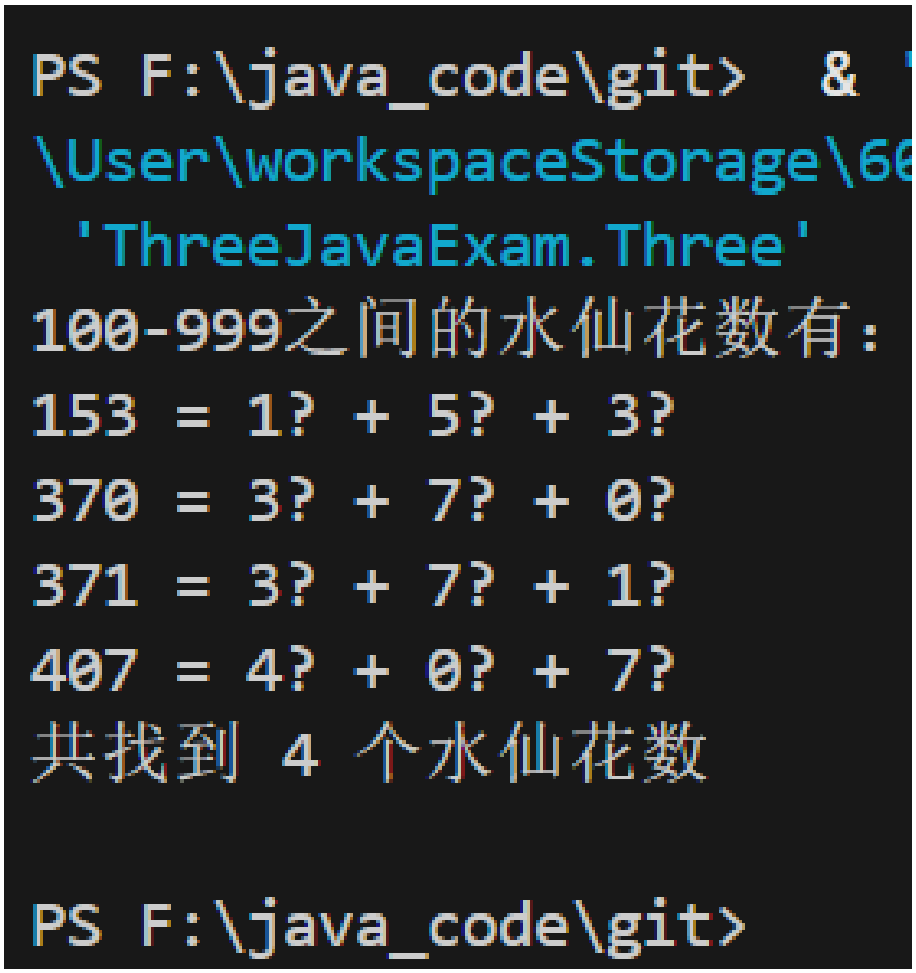

图 6: 运行结果

【第三个实验具体内容】

流程图

图 7: 流程图

部分源代码

```java
// Three.java
public void findAllOptimized() {
    results.clear();

    // Optimization for 3-digit numbers: iterate over each
       digit directly
    for (int hundreds = 1; hundreds <= 9; hundreds++) {
        for (int tens = 0; tens <= 9; tens++) {
            for (int ones = 0; ones <= 9; ones++) {
                int num = hundreds * 100 + tens * 10 + ones;
                if (num >= start && num <= end) {
                    int sum = hundreds * hundreds * hundreds
                            + tens * tens * tens
                            + ones * ones * ones;
                    if (sum == num) {
                        results.add(num);
                    }
                }
            }
        }
    }
}
public static void main(String[] args) {
    // Function test
    Three finder = new Three(100, 999);
    finder.findAllOptimized();
    finder.printResults();
}
```

图 8: ThreeQuestion 部分源代码

实验过程与结果

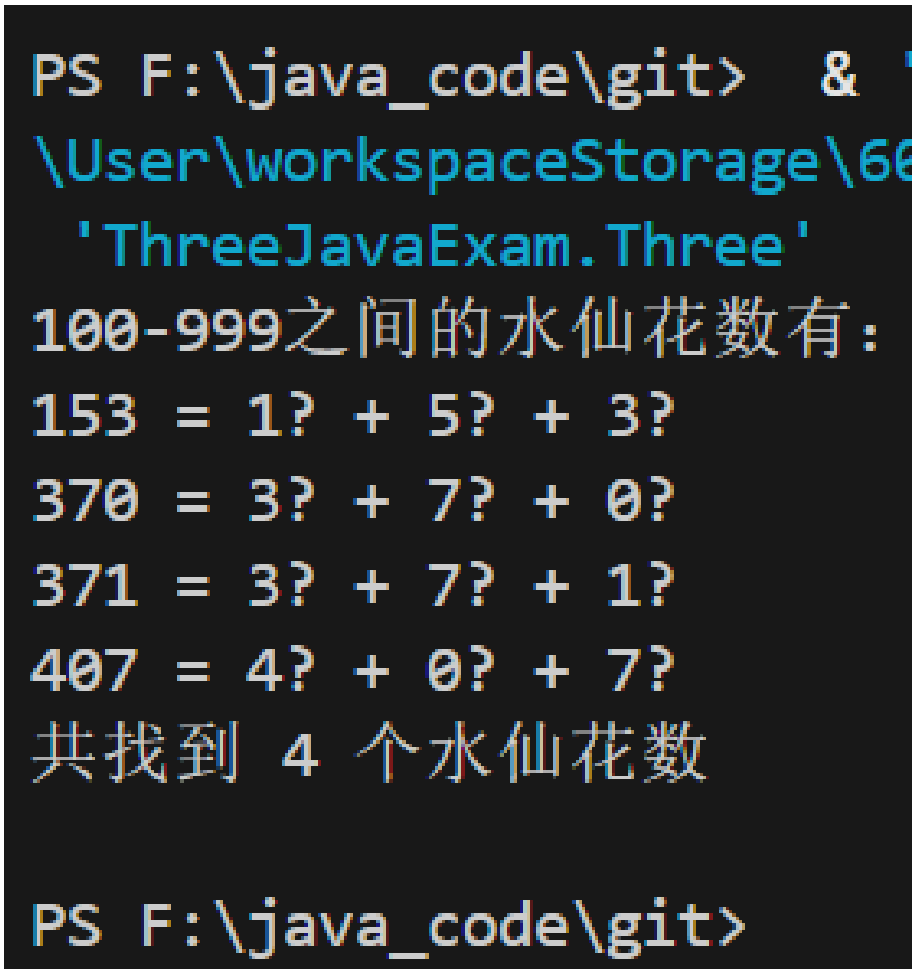

图 9: 运行结果

【第四个实验具体内容】

流程图

图 10: 流程图

部分源代码

```java
// FourQusetion.java
public void solveWithPruning() {
    solutions.clear();
    // Analyze ones place: Z + Z = ?2 (the ones digit of the
       result is 2)
    // Possible cases: Z = 1 (2), Z = 6 (12 requires carry)
    int[] possibleZ = findPossibleZ();
    for (int z : possibleZ) {
        int carry1 = (z + z) / 10; // carry from ones place
        // Analyze tens place: Y + Z + carry1 = ?3 (the tens
           digit is 3)
        int[] possibleY = findPossibleY(z, carry1);
        for (int y : possibleY) {
            int carry2 = (y + z + carry1) / 10; // carry from
                tens place
            // Analyze hundreds place: X + Y + carry2 = 5
            int x = 5 - y - carry2;
            if (x >= 1 && x <= 9) {
                Solution sol = new Solution(x, y, z);
                if (sol.getSum() == targetSum) {
                    solutions.add(sol);
                }
            }
        }
    }
}
```

图 11: FourQuestion 部分源代码

实验过程与结果

```
===== 实验4: 数字谜题求解器 =====
求解: XYZ + YZZ = 532

找到 1 个解:
解1: X=3, Y=2, Z=1  →  321 + 211 = 532
===== 解的验证 =====
验证: 321 + 211 = 532 ?
```

图 12: 运行结果

【第五个实验具体内容】

流程图

图 13: 流程图

部分源代码

```java
// FiveQusetion.java
public void solveOneLoop() {
    solutions.clear();
    // Only need to enumerate the number of hens
    for (int hen = 0; hen <= 20; hen++) {
        if ((100 - 7 * hen) % 4 == 0) {
            int rooster = (100 - 7 * hen) / 4;
            int chick = 100 - hen - rooster;
            if (rooster >= 0 && chick >= 0) {
                PurchasePlan plan = new PurchasePlan(hen,
                    rooster, chick);
                if (plan.isValid()) {
                    solutions.add(plan);
                }
            }
        }
    }
}
public static void main(String[] args) {
    System.out.println("===== Experiment 5: Hundred-Chickens-
        for-Hundred-Coins Problem =====\n");
    Five solver = new Five();
    System.out.println("Method 3: Single-loop (Optimal)");
    solver.solveOneLoop();
    solver.printSolutions("Method 3: Single-loop (Optimal)");
         // Print detailed solutions
}
```

图 14: FiveQuestion 部分源代码

实验过程与结果

```
PS F:\java_code\git>
PS F:\java_code\git>  f:; cd 'f:\java_code\git'; & 'E:\java8_s\bin\j
\ti_st\AppData\Roaming\Code\User\workspaceStorage\60c6439be9e3dc2adb
va\jdt_ws\git_30b76c12\bin' 'ThreeJavaExam.Five'
===== 实验5：百钱买百鸡问题 =====

【方法1：一层循环（最优）】
方法3：一层循环（最优）：
方案1：母鸡= 0只，公鸡=25只，小鸡=75只  (总价=100分，总数=100只)
方案2：母鸡= 4只，公鸡=18只，小鸡=78只  (总价=100分，总数=100只)
方案3：母鸡= 8只，公鸡=11只，小鸡=81只  (总价=100分，总数=100只)
方案4：母鸡=12只，公鸡= 4只，小鸡=84只  (总价=100分，总数=100只)
共有 4 种买法
```

图 15: 运行结果

【第六个实验具体内容】

流程图

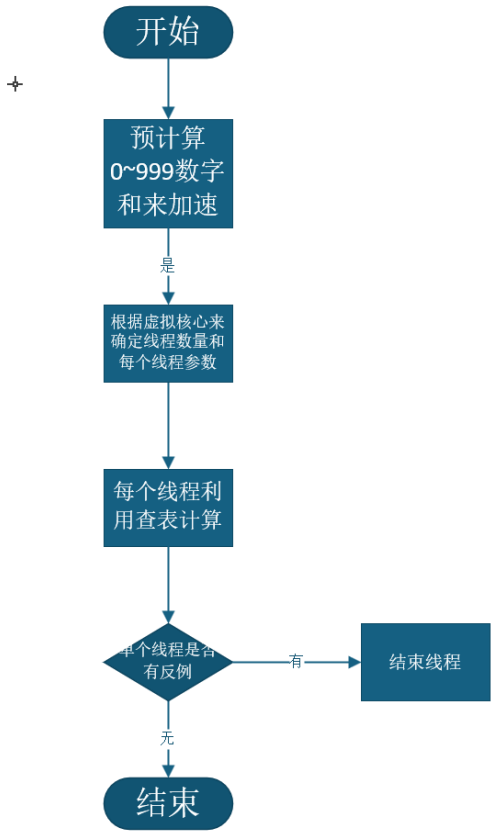

图 16: 流程图

部分源代码

```java
// Six.java
public static void main(String[] args) throws
    InterruptedException {
     int threads = Runtime.getRuntime().availableProcessors()
        * 2;
     long range = Integer.MAX_VALUE, step = range / threads;
     AtomicBoolean found = new AtomicBoolean(false);
     Thread[] ts = new Thread[threads];
     long start = System.nanoTime();

     for (int t = 0; t < threads; t++) {
         long a = t * step, b = (t == threads - 1) ? range + 1
             : (t + 1) * step;
         int id = t;
         ts[t] = new Thread(() -> {
             for (long n = a; n < b && !found.get(); n++) {
                 int x = (int) n, mod9 = x % 9, sum = fastSum(
                    x);
                 if (sum % 9 == 0 && mod9 != 0 && found.
                    compareAndSet(false, true))
                     System.out.printf("Thread %d found
                        counterexample: n=%d%n", id, x);
             }
         });
         ts[t].start();
     }
     for (Thread t : ts) t.join();
     System.out.printf("Done in %.3f s%n", (System.nanoTime()
        - start) / 1e9);
}
```

图 17: SixQuestion 部分源代码

实验过程与结果

极限的情况下可以达到0.7s内完成，我觉得大部分功劳是我cpu核心多，而且单独划出24G的内存给虚拟机使用，应该用时间复杂度来衡量优化的效果，单纯的时间不太好说。

这道题的最初使用的32线程速度在1.4s左右，后来在思考cpu核心的时候突然想到英特尔的超线程与AMD的并发线程，改用虚拟核心（不在单核单线程）。

还有一个优化是我突然想到java自动优化的常量池，我也搞了一个预计算池，直接查表来计算提高了不少。预计开启XTU之后可以突破0.5s，后续有时间我去尝试一下。

还有一个可以提高的部分是我只是开了单核双线程，实际上考率到线程的上下开销的问题，应该合理调整线程数量，这个得自己一个个测试，后续也可以调整好线程数量后进一步提高性能。

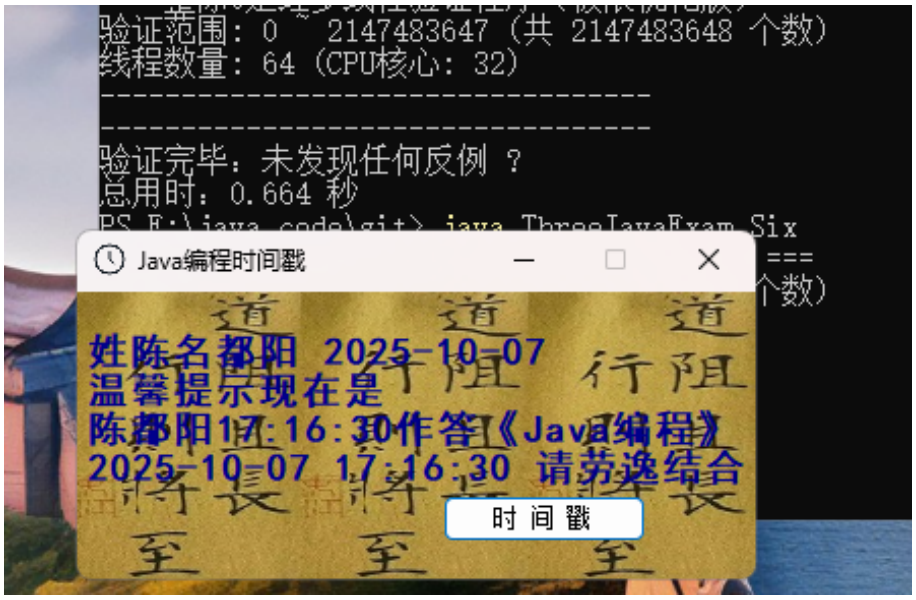

图 18: 运行结果

【第七个实验具体内容】

流程图

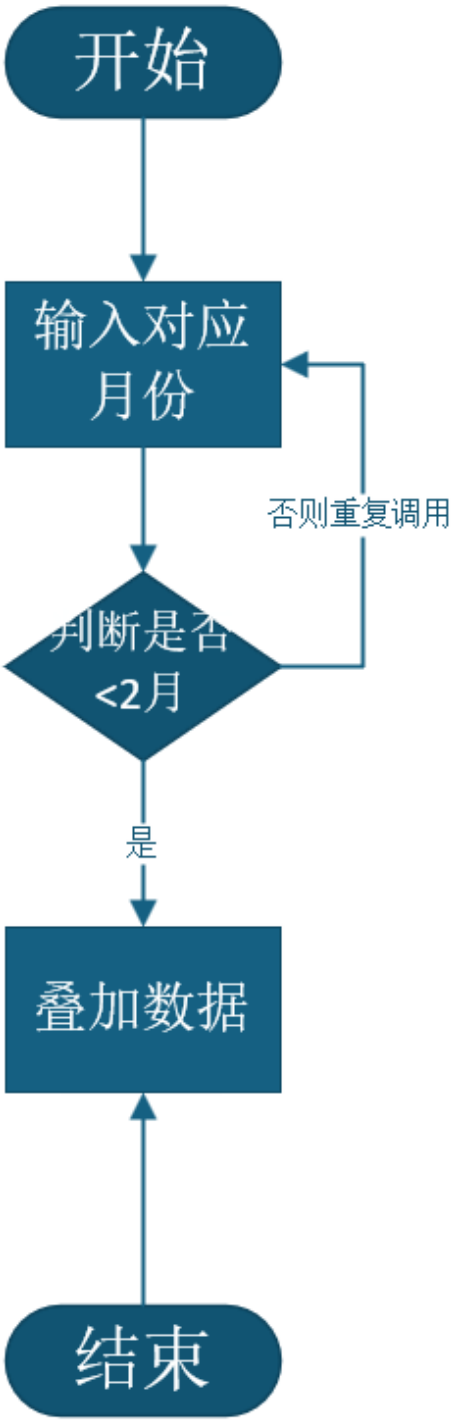

图 19: 流程图

部分源代码

```java
// Seven.java
public static long fibMemo(int n, long[] memo) {
    if (n <= 2) return 1L;
    if (memo[n] != 0) return memo[n];
    memo[n] = fibMemo(n - 1, memo) + fibMemo(n - 2, memo);
    return memo[n];
}
public static void main(String[] args) {
    int months = 24; // 24 months = 2 years
    long[] memo = new long[months + 1];
    long pairs = fibMemo(months, memo);          // Number of
       rabbit pairs
    long rabbits = pairs * 2;                    // Total
       number of individual rabbits
    System.out.println("After " + months + " months (2 years)
       :");
    System.out.println("Number of rabbit pairs = " + pairs);
    System.out.println("Total individual rabbits = " +
       rabbits);
}
```

图 20: SevenQuestion 部分源代码

实验过程与结果

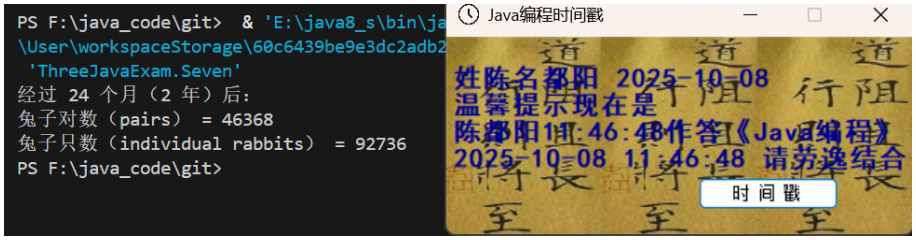

图 21: 运行结果

【第八个实验具体内容】

流程图

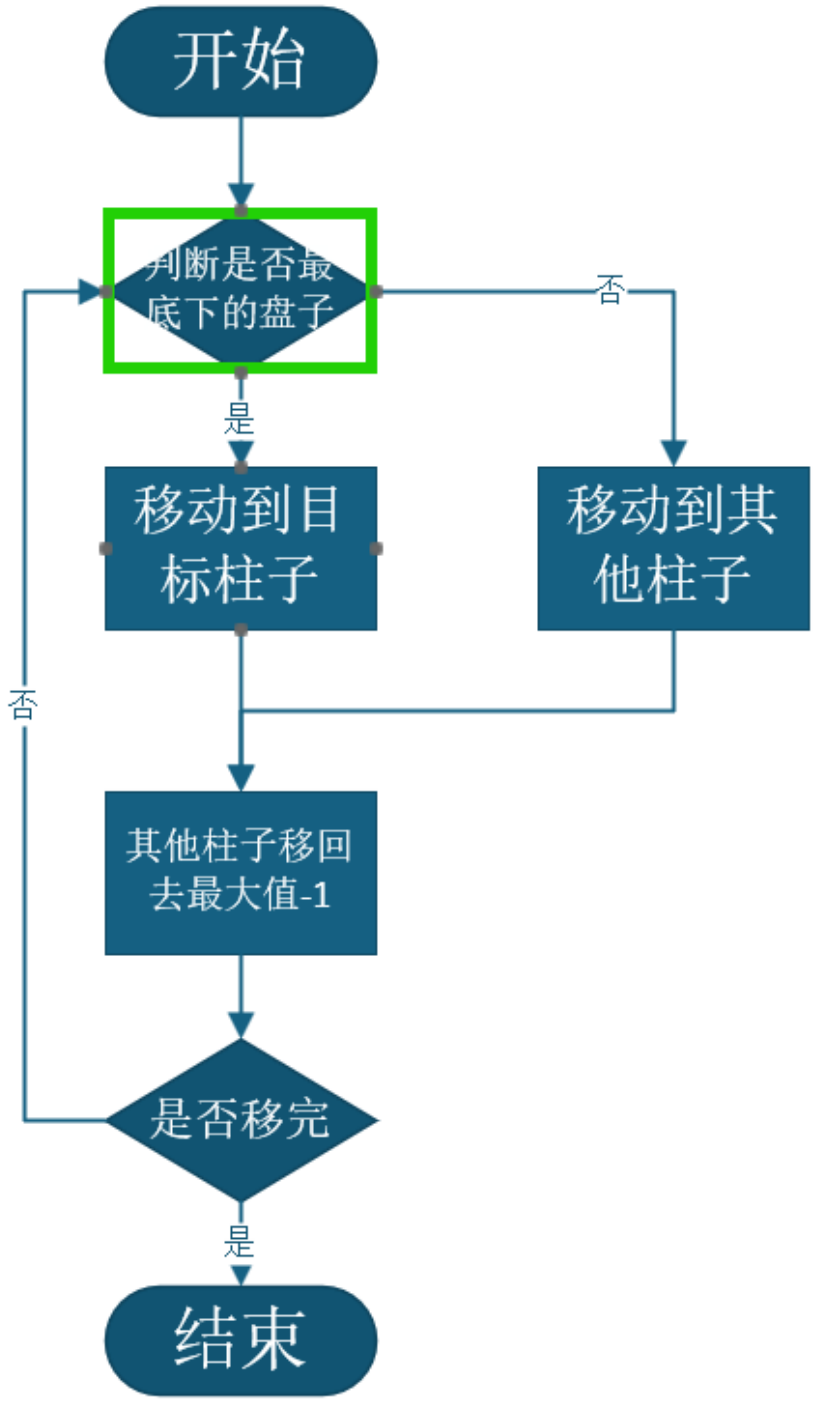

图 22: 流程图

部分源代码

```java
// Enight.java
public static void hanoi(int n, char from, char aux, char to)
    {
    if (n == 1) {
        System.out.println("Move disk 1: " + from + " -> " +
           to);
    } else {
        hanoi(n - 1, from, to, aux);
        System.out.println("Move disk " + n + ": " + from + "
            -> " + to);
        hanoi(n - 1, aux, from, to);
        }
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the number of disks n: ");
    int n = scanner.nextInt();
    System.out.println("\nSteps to solve the Tower of Hanoi:"
       );
    hanoi(n, 'A', 'B', 'C');
    long steps = (long) Math.pow(2, n) - 1;
    System.out.println("\nTotal moves required: " + steps);
    scanner.close();
}
```

图 23: FiveQuestion 部分源代码

实验过程与结果

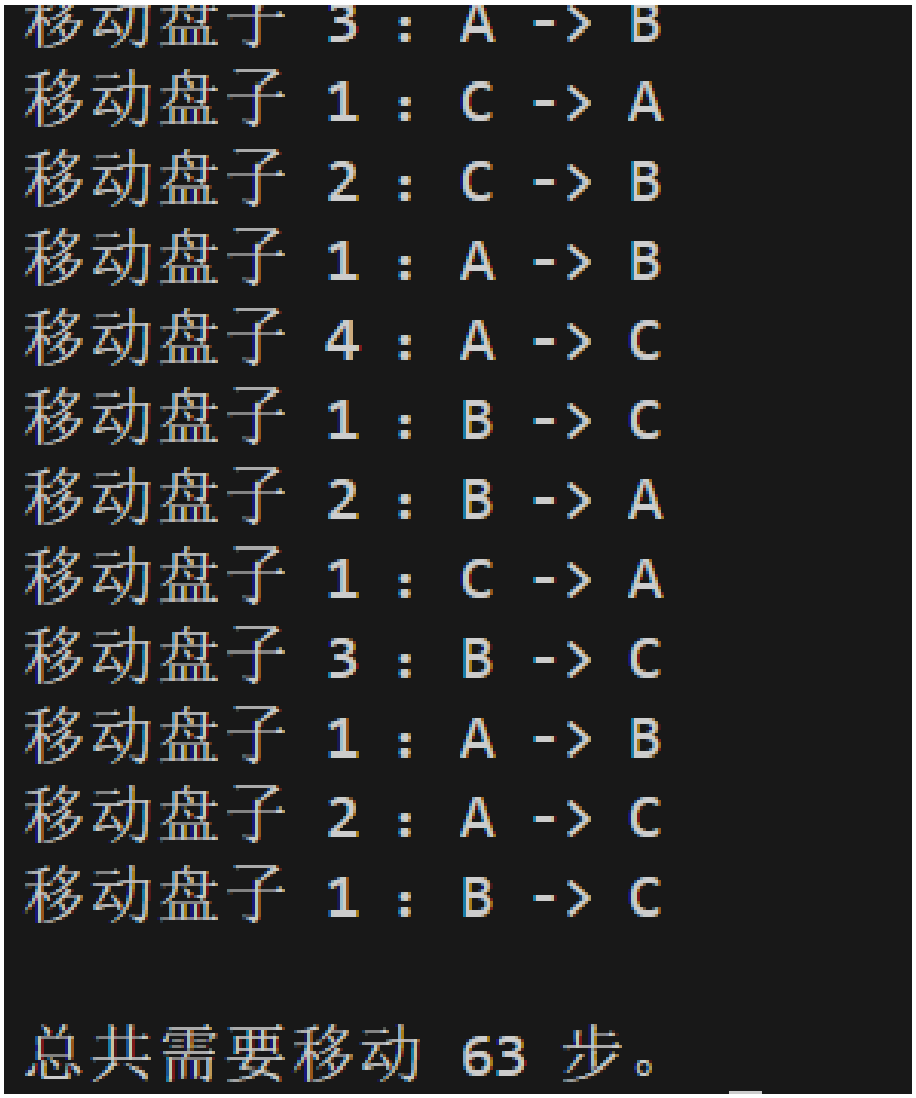

图 24: 运行结果

【实验心得】

本次实验主要花费时间在第六个实验的优化上，其他实验都比较简单，主要是考察对基本算法的理解与实现。

第六个实验多线程刚开始使用的是从C转过来的代码，刚开始使用的是pthread来管理多线程，后来用着用着老是在3S左右下不来，扒拉了一下数据手册，发现Java的线程管理更简单，直接使用Thread类就可以了。

另外一个优化是我突然想到java自动优化的常量池，我也搞了一个预计算池，直接查表来计算提高了不少。

我感觉用c语言来验证，套个Java的外壳来调用应该可以压缩一下，时间来不及了，先提交这版实验结果，后面有空再试试搓个C的子程序来试试。

而且给的数字2147483647刚好就是int的最大值，应该是有意为之，要是再大一些估计得搞两个int来存。

Java的使用感觉用来大批量流水化搓产品挺不错的，随便翻翻都能翻到自己想要的库，而且有很多现成的算法可以直接拿来用，省去了自己写算法的时间。至于需要优化提速的部分完全可以使用C语言来提速，实在不行还有汇编。不过Java的属性在方法里面调用还是有些小区别的

附录

【关键字索引】

abstract：声明抽象类或抽象方法。
assert：用于调试时的断言检查。
boolean：声明布尔类型变量（true 或 false）。
break：跳出当前循环或 switch 语句。
byte：声明 8 位有符号整数类型。
case：switch 语句中的分支标签。
catch：捕获异常。
char：声明字符类型（16 位 Unicode）。
class：声明一个类。
const：保留关键字，未使用。
continue：跳过当前循环的剩余部分并进入下一次循环。
default：switch 语句中的默认分支。
do：与 while 一起使用，构成 do-while 循环。
double：声明双精度浮点数类型。
else：if 语句的“否则”分支。
enum：定义枚举类型。
extends：表明一个类继承自另一个类。
final：声明常量、不可继承的类或不可重写的方法。
finally：定义在异常处理后一定会执行的代码块。
float：声明单精度浮点数类型。
for：定义 for 循环结构。
goto：保留关键字，未使用。
if：条件语句的起始关键字。
implements：声明一个类实现接口。
import：导入包中的类或接口。
instanceof：测试对象是否为某个类的实例。
int：声明整数类型（32 位）。
interface：定义接口。
long：声明长整型（64 位）。
native：声明一个本地方法（由其他语言实现）。
new：创建新对象实例。
null：表示空引用。
package：定义类所在的包。
private：声明私有访问权限。
protected：声明受保护访问权限。
public：声明公有访问权限。

return：从方法返回结果。
short：声明 16 位整数类型。
static：声明静态成员。
strictfp：限制浮点计算的精度和舍入。
super：引用父类的成员或构造方法。
switch：多分支选择语句。
synchronized：声明同步方法或代码块。
this：引用当前对象。
throw：抛出异常。
throws：声明可能抛出的异常类型。
transient：声明不被序列化的成员变量。
try：定义可能抛出异常的代码块。
void：声明方法无返回值。
volatile：声明变量在多线程中保持可见性。
while：定义 while 循环结构。
true、false、null：常量值（非严格意义关键字，但为保留字）。

【代码附录】

```java
// One.java
package ThreeJavaExam;

public class One {
    private int start;       // start range
    private int end;         // end range
    private int divisor;     // divisor
    private int sum;         // calculation result

    public One(int start, int end, int divisor) {
        this.start = start;
        this.end = end;
        this.divisor = divisor;
        this.sum = 0;
    }

    public void calculateByFormula() {
        // Find the first and last multiples in the range
        int firstMultiple = (start % divisor == 0) ? start :
           start + (divisor - start % divisor);
        int lastMultiple = end - (end % divisor);

        // If there are no multiples
        if (firstMultiple > end) {
            sum = 0;
            return;
        }
        int count = (lastMultiple - firstMultiple) / divisor
           + 1;
        sum = count * (firstMultiple + lastMultiple) / 2;
    }

    public void printResult(String method) {
        System.out.println(method + "48         ");
        System.out.println("The sum of all multiples of " +
           divisor + " between " + start + " and " + end + "
           is: " + sum);
    }

    public static void main(String[] args) {
```

```
        One calculator = new One(1, 2000, 3);
        calculator.calculateByFormula();
        calculator.printResult("By arithmetic sequence summation");
    }
}
```

```
// Two.java
/**
 * Experiment 2: Determine the number of days in a given year and month
 */
package ThreeJavaExam;

import java.util.Scanner;

public class Two {
    private int year;
    private int month;

    /**
     * Constructor
     */
    public Two(int year, int month) {
        this.year = year;
        this.month = month;
    }

    /**
     * Check if the year is a leap year
     */
    public boolean isLeapYear() {
        return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
    }


    /**
     * Optimized method using an array to store days
     */
    public int getDaysByArray() {
        if (month < 1 || month > 12) {
            return -1;
```

```
35          }
36
37          // Array storing days in each month (index 0 unused
                for direct month correspondence)
38          int[] daysInMonth = {0, 31, 28, 31, 30, 31, 30, 31,
                31, 30, 31, 30, 31};
39
40          // If February and leap year, return 29 days
41          if (month == 2 && isLeapYear()) {
42              return 29;
43          }
44
45          return daysInMonth[month];
46      }
47
48      /**
49       * Validate if the month is legal
50       */
51      public boolean isValidMonth() {
52          return month >= 1 && month <= 12;
53      }
54
55      /**
56       * Get the month name in Chinese numerals
57       */
58      public String getMonthName() {
59          return isValidMonth() ? monthNames[month] : "invalid"
                ;
60      }
61
62      /**
63       * Print detailed information
64       */
65      public void printDetails() {
66          if (!isValidMonth()) {
67              System.out.println("Error: Invalid month!");
68              return;
69          }
70
71          System.out.println("\n===== Date Information =====");
72          System.out.println("Year: " + year);
73          System.out.println("Is leap year: " + (isLeapYear() ?
                "Yes" : "No"));
```

```
74          System.out.println("Number of days: " +
                getDaysByArray() + " days");
75          System.out.println("==================\n");
76      }
77
78      // Getter and Setter methods
79      public int getYear() { return year; }
80      public int getMonth() { return month; }
81      public void setYear(int year) { this.year = year; }
82      public void setMonth(int month) { this.month = month; }
83
84      /**
85       * Main program entry
86       */
87      public static void main(String[] args) {
88          Scanner scanner = new Scanner(System.in);
89
90          System.out.println("===== Experiment 2: Date
                Calculator =====");
91          System.out.print("Please enter the year: ");
92          int year = scanner.nextInt();
93
94          System.out.print("Please enter the month (1-12): ");
95          int month = scanner.nextInt();
96
97          // Create date calculator object
98          Two calculator = new Two(year, month);
99
100         // Display detailed information
101         calculator.printDetails();
102
103         scanner.close();
104     }
105 }
```

```
1  // Three.java
2  /**
3   * Experiment 3: Narcissistic Number Finder
4   */
5  package ThreeJavaExam;
6
7  import java.util.ArrayList;
8  import java.util.List;
```

```java

public class Three {
    private int start;      // Search range start
    private int end;        // Search range end
    private List<Integer> results;  // Store found narcissistic numbers

    /**
     * Constructor
     */
    public Three(int start, int end) {
        this.start = start;
        this.end = end;
        this.results = new ArrayList<>();
    }

    /**
     * Find all narcissistic numbers in the range
     */
    public void findAllOptimized() {
        results.clear();

        // Optimization for 100-999: directly iterate through each digit
        for (int hundreds = 1; hundreds <= 9; hundreds++) {
            for (int tens = 0; tens <= 9; tens++) {
                for (int units = 0; units <= 9; units++) {
                    int num = hundreds * 100 + tens * 10 + units;
                    if (num >= start && num <= end) {
                        int sum = hundreds * hundreds * hundreds + tens * tens * tens + units * units * units;
                        if (sum == num) {
                            results.add(num);
                        }
                    }
                }
            }
        }
    }

    /**
```

```
47          * Get the digit decomposition of a number
48          */
49         public int[] getDigits(int num) {
50             int units = num % 10;
51             int tens = (num / 10) % 10;
52             int hundreds = num / 100;
53             return new int[]{hundreds, tens, units};
54         }
55 
56         /**
57          * Print the results
58          */
59         public void printResults() {
60             if (results.isEmpty()) {
61                 System.out.println("There are no narcissistic
                       numbers between " + start + " and " + end);
62                 return;
63             }
64 
65             System.out.println("Narcissistic numbers between " +
                   start + " and " + end + " are:");
66             for (int num : results) {
67                 int[] digits = getDigits(num);
68                 System.out.printf("%d = %d  + %d  + %d  %n",
69                                   num, digits[0], digits[1], digits
                                       [2]);
70             }
71             System.out.println("Total found: " + results.size() +
                   " narcissistic numbers\n");
72         }
73 
74         public static void main(String[] args) {
75             // Function test
76             Three finder = new Three(100, 999);
77             finder.findAllOptimized();
78             finder.printResults();
79 
80         }
81 }
```

```
1  // Four.java
2  /**
3   * Experiment 4: Number Puzzle Solver
```

```
 */
package ThreeJavaExam;

import java.util.ArrayList;
import java.util.List;


public class Four {

    /**
     * Inner Class: Represents a solution
     */
    public static class Solution {
        private int x, y, z;
        private int xyz, yzz;

        public Solution(int x, int y, int z) {
            this.x = x;
            this.y = y;
            this.z = z;
            this.xyz = x * 100 + y * 10 + z;
            this.yzz = y * 100 + z * 10 + z;
        }

        public int getX() { return x; }
        public int getY() { return y; }
        public int getZ() { return z; }
        public int getXYZ() { return xyz; }
        public int getYZZ() { return yzz; }
        public int getSum() { return xyz + yzz; }

        public String toString() {
            return String.format("X=%d, Y=%d, Z=%d  48
                    %d + %d = %d",
                                 x, y, z, xyz, yzz, getSum());
        }
    }

    private int targetSum;
    private List<Solution> solutions;

    /**
     * Constructor
```

```
46          */
47         public Four(int targetSum) {
48             this.targetSum = targetSum;
49             this.solutions = new ArrayList<>();
50         }
51 
52         /**
53          * Method 1: Solve using carry relationships of units,
               tens, and hundreds places
54          */
55         public void solveWithPruning() {
56             solutions.clear();
57 
58             // Analyze units place: Z + Z = ?2 (units digit of
                   result is 2)
59             // Possible values: Z=1 (sum=2), Z=6 (sum=12 with
                   carry)
60             int[] possibleZ = findPossibleZ();
61 
62             for (int z : possibleZ) {
63                 int carry1 = (z + z) / 10;  // Carry from units
                       place
64 
65                 // Analyze tens place: Y + Z + carry1 = ?3 (tens
                       digit of result is 3)
66                 int[] possibleY = findPossibleY(z, carry1);
67 
68                 for (int y : possibleY) {
69                     int carry2 = (y + z + carry1) / 10;  // Carry
                           from tens place
70 
71                     // Analyze hundreds place: X + Y + carry2 = 5
72                     int x = 5 - y - carry2;
73 
74                     if (x >= 1 && x <= 9) {
75                         Solution sol = new Solution(x, y, z);
76                         if (sol.getSum() == targetSum) {
77                             solutions.add(sol);
78                         }
79                     }
80                 }
81             }
82         }
```

```
83
84      private int[] findPossibleZ() {
85          List<Integer> possible = new ArrayList<>();
86          for (int z = 0; z <= 9; z++) {
87              if ((z + z) % 10 == 2) {  // Units digit of sum
                    is 2 after adding Z + Z
88                  possible.add(z);
89              }
90          }
91          return possible.stream().mapToInt(i -> i).toArray();
92      }
93
94      private int[] findPossibleY(int z, int carry1) {
95          List<Integer> possible = new ArrayList<>();
96          for (int y = 1; y <= 9; y++) {
97              if ((y + z + carry1) % 10 == 3) {  // Tens digit
                    of sum is 3 after calculation
98                  possible.add(y);
99              }
100         }
101         return possible.stream().mapToInt(i -> i).toArray();
102     }
103
104     /**
105      * Print all solutions
106      */
107     public void printSolutions() {
108         if (solutions.isEmpty()) {
109             System.out.println("No solutions found!");
110             return;
111         }
112
113         System.out.println("Found " + solutions.size() + "
                solution(s):");
114         for (int i = 0; i < solutions.size(); i++) {
115             System.out.println("Solution " + (i + 1) + ": " +
                    solutions.get(i));
116         }
117     }
118
119     /**
120      * Get list of solutions
121      */
```

```
122     public List<Solution> getSolutions() {
123         return new ArrayList<>(solutions);
124     }
125 
126     public static void main(String[] args) {
127         System.out.println("===== Experiment 4: Number Puzzle
                Solver =====");
128         System.out.println("Solving: XYZ + YZZ = 532\n");
129 
130         Four solver = new Four(532);
131 
132         // Solve using optimized method
133         solver.solveWithPruning();
134         solver.printSolutions();
135 
136         // Verify correctness of solutions
137         System.out.println("===== Solution Verification =====
               ");
138         for (Solution sol : solver.getSolutions()) {
139             System.out.printf("Verification: %d + %d = %d %s%
                   n",
140                               sol.getXYZ(), sol.getYZZ(), sol.
                                  getSum(),
141                               sol.getSum() == 532 ? "48      :
"□");
```

```
1  // Five.java
2  /**
3   * Experiment 5: Hundred Coins for Hundred Chickens Problem
4   */
5  package ThreeJavaExam;
6 
7  import java.util.ArrayList;
8  import java.util.List;
9 
10 
11 public class Five {
12 
13     /**
14      * Inner Class: Represents a purchase plan
15      */
16     public static class PurchasePlan {
17         private int hen;         // Number of hens
```

```
18          private int rooster;     // Number of roosters
19          private int chick;       // Number of chicks
20
21          // Fixed prices (unit: coins)
22          private static final int HEN_PRICE = 5;
23          private static final int ROOSTER_PRICE = 3;
24          private static final int CHICK_GROUP_PRICE = 1;  // 3
                chicks cost 1 coin
25
26          public PurchasePlan(int hen, int rooster, int chick)
               {
27              this.hen = hen;
28              this.rooster = rooster;
29              this.chick = chick;
30          }
31
32          // Getter methods
33          public int getHen() { return hen; }
34          public int getRooster() { return rooster; }
35          public int getChick() { return chick; }
36
37          /**
38           * Calculate total number of chickens
39           */
40          public int getTotalCount() {
41              return hen + rooster + chick;
42          }
43
44          /**
45           * Calculate total cost (unit: coins)
46           */
47          public int getTotalPrice() {
48              return hen * HEN_PRICE + rooster * ROOSTER_PRICE
                    + chick / 3;
49          }
50
51          /**
52           * Check if the purchase plan is valid
53           * Conditions: total count = 100, total cost = 100,
                chicks count is divisible by 3
54           */
55          public boolean isValid() {
56              return getTotalCount() == 100 &&
```

```
57                      getTotalPrice() == 100 &&
58                      chick % 3 == 0;
59          }
60
61          /**
62           * Format plan information as string
63           */
64          public String toString() {
65              return String.format("Hens=%2d, Roosters=%2d,
                    Chicks=%2d  (Total price=%3d coins, Total
                    count=%3d)",
66                                   hen, rooster, chick,
                                        getTotalPrice(),
                                        getTotalCount());
67          }
68      }
69
70      private List<PurchasePlan> solutions;  // Store all valid
            purchase plans
71
72      /**
73       * Constructor: Initialize solution list
74       */
75      public Five() {
76          this.solutions = new ArrayList<>();
77      }
78
79      /**
80       * Basic solving method: Triple loop (brute force)
81       * Iterate all possible counts of hens, roosters and
            chicks
82       */
83      public void solveTripleLoop() {
84          solutions.clear();
85
86          // Hens can't exceed 20 (100/5)
87          for (int hen = 0; hen <= 20; hen++) {
88              // Roosters can't exceed 33 (100/3)
89              for (int rooster = 0; rooster <= 33; rooster++) {
90                  // Chicks count is determined (total 100)
91                  int chick = 100 - hen - rooster;
92
93                  // Chicks must be non-negative and divisible
```

```
                    by 3
94                  if (chick >= 0 && chick % 3 == 0) {
95                      PurchasePlan plan = new PurchasePlan(hen,
                            rooster, chick);
96                      if (plan.isValid()) {
97                          solutions.add(plan);
98                      }
99                  }
100             }
101         }
102     }
103
104     /**
105      * Optimized method: Double loop
106      * Reduce one loop by calculating chicks count directly
107      */
108     public void solveDoubleLoop() {
109         solutions.clear();
110
111         for (int hen = 0; hen <= 20; hen++) {
112             // Calculate maximum possible roosters based on
                    remaining money
113             int maxRooster = (100 - hen * HEN_PRICE) /
                    ROOSTER_PRICE;
114             for (int rooster = 0; rooster <= maxRooster;
                    rooster++) {
115                 int chick = 100 - hen - rooster;
116                 if (chick >= 0 && chick % 3 == 0) {
117                     PurchasePlan plan = new PurchasePlan(hen,
                            rooster, chick);
118                     if (plan.isValid()) {
119                         solutions.add(plan);
120                     }
121                 }
122             }
123         }
124     }
125
126     /**
127      * Optimal solving method: Single loop
128      * Derived from mathematical equations to minimize
            computation
129      */
```

```
130     public void solveOneLoop() {
131         solutions.clear();
132 
133         // Only iterate through possible hen counts (0-20)
134         for (int hen = 0; hen <= 20; hen++) {
135             // Check if rooster count can be an integer (from
                equation derivation)
136             if ((100 - 7 * hen) % 4 == 0) {
137                 int rooster = (100 - 7 * hen) / 4;
138                 int chick = 100 - hen - rooster;
139 
140                 // Ensure non-negative counts
141                 if (rooster >= 0 && chick >= 0) {
142                     PurchasePlan plan = new PurchasePlan(hen,
                            rooster, chick);
143                     if (plan.isValid()) {
144                         solutions.add(plan);
145                     }
146                 }
147             }
148         }
149     }
150 
151     /**
152      * Print all valid solutions
153      *  methodName Name of the solving method to display
154      */
155     public void printSolutions(String methodName) {
156         if (solutions.isEmpty()) {
157             System.out.println("No valid solutions found!");
158             return;
159         }
160 
161         System.out.println(methodName + ":");
162         for (int i = 0; i < solutions.size(); i++) {
163             System.out.printf("Solution %d: %s%n", i + 1,
                    solutions.get(i));
164         }
165         System.out.println("Total " + solutions.size() + "
               solutions\n");
166     }
167 
168     /**
```

```
169         * Get copy of solution list (avoid external modification
                )
170         */
171        public List<PurchasePlan> getSolutions() {
172            return new ArrayList<>(solutions);
173        }
174
175        /**
176         * Main method: Compare different solving methods
177         */
178        public static void main(String[] args) {
179            System.out.println("===== Experiment 5: Hundred Coins
                    for Hundred Chickens Problem =====\n");
180
181            Five solver = new Five();
182
183            // Method 1: Triple loop (brute force)
184            System.out.println("[Method 1: Triple loop (brute
                   force)]");
185            solver.solveTripleLoop();
186            solver.printSolutions("Triple loop results");
187
188            // Method 2: Double loop (optimized)
189            System.out.println("[Method 2: Double loop (optimized
                   )]");
190            solver.solveDoubleLoop();
191            solver.printSolutions("Double loop results");
192
193            // Method 3: Single loop (most efficient)
194            System.out.println("[Method 3: Single loop (optimal)]
                   ");
195            solver.solveOneLoop();
196            solver.printSolutions("Single loop results");
197        }
198    }
```

```
1   // Six.java
2   package ThreeJavaExam;
3
4   import java.util.concurrent.atomic.AtomicBoolean;
5
6   /**
7    * Experiment 6: Verify the theorem of divisibility by 9
```

```java
 */
public class Six {
    // Precompute the sum of digits for 0-999
    private static final int[] DIGIT_SUM_CACHE = new int
        [1000];

    static {
        for (int i = 0; i < 1000; i++) {
            int sum = 0;
            int n = i;
            while (n > 0) {
                sum += n % 10;
                n /= 10;
            }
            DIGIT_SUM_CACHE[i] = sum;
        }
    }

    public static void main(String[] args) throws
        InterruptedException {
        // Number of threads and range
        final int THREAD_COUNT = Runtime.getRuntime().
            availableProcessors() * 2; // Dynamically adjust
        final long RANGE = (long) Integer.MAX_VALUE;
        final long STEP = RANGE / THREAD_COUNT;

        System.out.println("Verification range: 0 ~ " + RANGE
            + " (total " + (RANGE + 1) + " numbers)");
        System.out.println("Number of threads: " +
            THREAD_COUNT + " (CPU cores: " + Runtime.
            getRuntime().availableProcessors() + ")");

        AtomicBoolean found = new AtomicBoolean(false);
        Thread[] threads = new Thread[THREAD_COUNT];
        long startTime = System.nanoTime();

        for (int t = 0; t < THREAD_COUNT; t++) {
            final int threadId = t;
            final long start = t * STEP;
            final long end = (t == THREAD_COUNT - 1) ? RANGE
                + 1 : (t + 1) * STEP;  // +1 to include
                maximum value

```

```
43              threads[t] = new Thread(() -> {
44                  // Core mathematical principle: n % 9 ==
                        digitSum(n) % 9
45                  // So we only need to verify if this equation
                        always holds
46                  for (long n = start; n < end && !found.get();
                        n++) {
47                      int num = (int) n;
48
49                      // Skip obviously valid cases (multiples
                            of 9)
50                      int mod9 = num % 9;
51                      if (mod9 == 0) continue; // For multiples
                            of 9, sum of digits must be a
                            multiple of 9
52
53                      // Quickly calculate sum of digits (using
                            lookup table)
54                      int sum = fastDigitSum(num);
55                      int sumMod9 = sum % 9;
56
57                      // Check if the theorem is violated
58                      if (sumMod9 == 0 && mod9 != 0) {
59                          if (found.compareAndSet(false, true))
                                {
60                              System.out.printf("Thread %d
                                    found a counterexample! n=%d
                                    sum of digits=%d (sum%%9==0,
                                    but n%%9=%d)%n",
61                                      threadId, num, sum, mod9)
                                          ;
62                          }
63                          return;
64                      }
65                  }
66              });
67              threads[t].start();
68          }
69
70          // Wait for all threads to complete
71          for (Thread thread : threads) {
72              thread.join();
73          }
```

```
74
75          long time = System.nanoTime() - startTime;
76          System.out.println("
               ----------------------------------");
77          if (found.get())
78              System.out.println("Verification terminated early
                   : counterexample found ");
79          else
80              System.out.println("Verification completed: no
                   counterexamples found ");
81          System.out.printf("Total time: %.3f seconds%n", time
               / 1_000_000_000.0);
82      }
83
84      /**
85       * Calculate using precomputed table
86       */
87      private static int fastDigitSum(int n) {
88          if (n < 1000) {
89              return DIGIT_SUM_CACHE[n];
90          }
91
92          int sum = 0;
93
94          // Process in segments: handle 3 digits each time
95          while (n >= 1000) {
96              int last3 = n % 1000;
97              sum += DIGIT_SUM_CACHE[last3];
98              n /= 1000;
99          }
100
101         // Process remaining part
102         sum += DIGIT_SUM_CACHE[n];
103
104         return sum;
105     }
106 }
```

```
1  // Seven.java
2  package ThreeJavaExam;
3
4  public class Seven {
5
```

```
6       /**
7        * Calculates Fibonacci number using memoization (
             recursive approach)
8        */
9       public static long fibMemo(int n, long[] memo) {
10          // Base case: 1 pair of rabbits in the 1st and 2nd
               months
11          if (n <= 2) return 1L;
12          // Return cached value if already computed
13          if (memo[n] != 0) return memo[n];
14          // Recursively calculate and store result in memo
               array
15          memo[n] = fibMemo(n - 1, memo) + fibMemo(n - 2, memo)
               ;
16          return memo[n];
17      }
18
19      public static void main(String[] args) {
20          int months = 24; // Target period: 24 months
21
22          // Memoization array: index represents month, value
               represents rabbit pairs
23          long[] memo = new long[months + 1];
24          long rabbitPairs = fibMemo(months, memo);          //
               Calculate number of rabbit pairs
25          long rabbitIndividuals = rabbitPairs * 2;          //
               Convert pairs to individuals (1 pair = 2 rabbits)
26
27          System.out.println("After " + months + " months (2
               years):");
28          System.out.println("Number of rabbit pairs = " +
               rabbitPairs);
29          System.out.println("Number of individual rabbits = "
               + rabbitIndividuals);
30      }
31  }
```

```
1   // Eight.java
2   package ThreeJavaExam;
3
4   import java.util.Scanner;
5
6   public class Enight {
```

```
7
8       /**
9        * Implement Tower of Hanoi recursively
10       */
11      public static void hanoi(int n, char from, char aux, char
            to) {
12          if (n == 1) {
13              System.out.println("Move disk 1: " + from + " ->
                    " + to);
14          } else {
15              // First move the top n-1 disks from source to
                    auxiliary
16              hanoi(n - 1, from, to, aux);
17              // Then move the largest disk from source to
                    target
18              System.out.println("Move disk " + n + ": " + from
                     + " -> " + to);
19              // Finally move the n-1 disks from auxiliary to
                    target
20              hanoi(n - 1, aux, from, to);
21          }
22      }
23
24      /**
25       * Main function: read input and call recursive function
26       */
27      public static void main(String[] args) {
28          Scanner scanner = new Scanner(System.in);
29
30          System.out.print("Please enter the number of disks n:
                ");
31          int n = scanner.nextInt();
32
33          System.out.println("\nTower of Hanoi movement steps:"
                );
34          hanoi(n, 'A', 'B', 'C');
35
36          // Total steps formula: 2^n - 1
37          long steps = (long) Math.pow(2, n) - 1;
38          System.out.println("\nTotal steps required: " + steps
                + ".");
39
40          scanner.close();
```

```
41          }
42  }
```