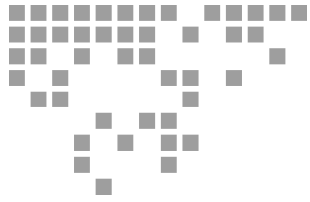

Chapter 14 第 14 章

算法面试题

在算法面试中，主要的面试内容分为基础算法和思维扩展算法两部分。基础算法主要考查面试者的基础是否牢固；思维扩展算法主要考查面试者分析、解决问题和临场反应等方面的能力。

基础算法都是比较常见的算法，描述规范、熟练程度非常重要，只要能够按照要求实现即可，思维扩展算法首先需要面试者建立编程模型，然后使用代码编程并给出答案。本章分析了经常出现的算法，希望能够给读者带来帮助和启示。

14.1 基础算法

基础算法是算法的基本功，读者一定要熟练掌握。基础算法主要包括：字符串匹配、分解质因数、数字排列、数字组合等。

14.1.1 字符串匹配

考题题干：

判断字符串 a 是否包含字符串 b。这里称 a 为文本串，b 为模式串，如图 14-1 所示。

```
a = bcabcabcabbcabcabcabcabd
    ||||||||||/
b = bcabcabcabc
```

图 14-1 字符串匹配

考题分析：

如图 14-1 中“/”处两个字符匹配失败，如果模式串右移一个字符，从文本串第二个字符开始重新进行匹配，显然效率太低。该算法的精髓在于当文本串和模式串发生不匹配时，利

用模式串自身的特点，尽可能多地移动模式串，使之能够在文本串不匹配处继续进行匹配。发生不匹配时，模式串可以右移三次，在目标串不匹配处继续进行，如图 14-2 所示。

```
a=bcabcabcabbcabcabcabcabd
     |||||||/
b=   bcabcabcabc
```

图 14-2　字符串匹配

模式串 b=$b_1b_2b_3b_4\cdots b_n$，如果有 $b_1b_2b_3b_4\cdots b_k$-1 等于 b_j-(k-1)b_j-(k-2)b_j-(k-3)$\cdots b_j$-1，那么在进行匹配时，如果在 j 处与文本串 i 处匹配失败，可以将模式串右移 j-k 个字符，即直接将文本串 i 处的字符与 b_k 进行匹配。由于 $b_1b_2b_3b_4\cdots b_k$-1 等于 b_j-(k-1)b_j-(k-2)b_j-(k-3)$\cdots b_j$-1，显然模式串前 k-1 位的字符是与文本串 i 处前 k-1 位的字符是匹配的。

所以问题转化为求模式串各字符的 K 值，记作 $K(j)$，该值可以根据模式串自身求得。

参考代码如下：

```
import java.util.Scanner;

public class KMPAlgorithm
{

    /**
     * 判断是否匹配
     * @param target 目标文本串
     * @param mode 模式串
     * @return 匹配结果
     */
    public static boolean matchString(String target, String mode)
    {
        //为了和算法保持一致，使index从1开始，增加一个前缀
        String newTarget = "x" + target;
        String newMode = 'x' + mode;

        int[] K = calculateK(mode);

        int i = 1;
        int j = 1;

        while(i <= target.length() && j <= mode.length())
        {
            if (j == 0 || newTarget.charAt(i) == newMode.charAt(j))
            {
                i++;
                j++;
            }
            else
            {
                j = K[j];
            }
        }

        if (j > mode.length())
        {
```

```
            return true;
        }
        return false;
    }

     /*
     * 计算K值
     */
    private static int[] calculateK(String mode)
    {
        //为了和算法保持一致，使index从1开始，增加一个前缀
        String newMode = "x" + mode;
        int[] K = new int[newMode.length()];
        int i = 1;
        K[1] = 0;
        int j = 0;

        while(i < mode.length())
        {
            if (j == 0 || newMode.charAt(i) == newMode.charAt(j))
            {
                i++;
                j++;
                K[i] = j;
            }
            else
            {
                j = K[j];
            }
        }

        return K;
}

    /**
     * @param args
     */
    public static void main(String[] args)
    {
        String s1,s2;
        boolean b;
       Scanner input=new Scanner(System.in);

       System.out.println("请输入字符串1: ");
       s1=input.next();
       System.out.println("请输入字符串2: ");
       s2=input.next();

       b=KMPAlgorithm.matchString(s1, s2);

       System.out.println("匹配成功? "+b);

}

}
```

程序输出结果如图 14-3 所示。

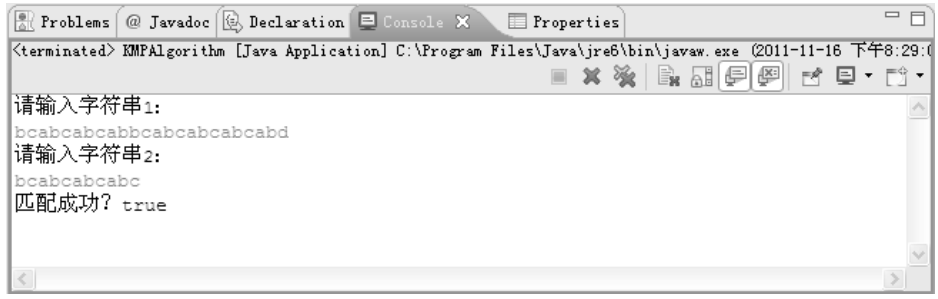

图 14-3　程序输出结果

14.1.2　哥德巴赫猜想的近似证明

考题题干：

哥德巴赫猜想即任何一个大于 2 的偶数都能表示为两个素数之和。编写一个 Java 程序，验证 1～200 内哥德巴赫猜想的正确性，近似证明哥德巴赫猜想。

考题分析：

可以应用枚举的方法列出 1～200 内的所有偶数。然后逐一验证每个偶数是否满足哥德巴赫猜想的论证。如果有一个不符合，就意味着哥德巴赫猜想不成立。

一个正偶数 m 可以表示成 m=1+(m－1)，m=2+(m－2)，m=3+(m－3)，……，m=m/2+m/2。由于 m/2 的后半部分和前半部分的结果是一样的，只是加数顺序不同，所以可以忽略。

参考代码：

```
public class Guess
{
   public static boolean isPrime(int i)
      {
         // 判断参数i是否是素数，是则返回true反之则返回false
         int n;
         boolean flag = true;
         if (1 == i)                    //1本身不是素数，因此需把这个特殊的数字抛出
            flag = false;

         for (n = 2; n <= i - 1; n++)  /* 判断i是否是素数的一个方法是看2～i－1之间是
         否有其因子（能被2整除），有则不是素数返回false，反之则返回true*/
            if (i % n == 0)
            {
               flag = false;
               break;
            }
         return flag;
      }

   public static boolean isGoldbach(int a)
      {                                 // 判断参数a是否符合哥德巴赫猜想
         int i;
         boolean flag = false;
```

```
            for (i = 1; i <= a / 2; i++)
            {
                if (isPrime(i) && isPrime(a - i))
                {    // 根据试题分析中的表达式，传入相关的两个参数
                    flag = true;
                    System.out.printf("%3d=%3d+%3d  ",a,i,(a - i));
                    break;
                     // 只要有一个符合条件的就可以退出循环，判断下一个偶数
                }
            }
            return flag;
        }

     public static boolean Testify_Guess(int low, int high)
        {
            // 判断1～100范围内的所有偶数是否符合哥德巴赫猜想,符合则返回true,反之则返回false
            int i, j = 0;
            boolean flag = true;
            for (i = low; i <= high; i++)
                if (i % 2 == 0 && i > 2)  //在1～200之间选取大于2的偶数进行猜想测试
                    if (isGoldbach(i))
                    {
                        j++;                //j用来控制输出格式 ，每行输出5个数据
                        if (j == 10)
                        {
                            System.out.println();
                            j = 0;
                        }
                    }
                    else
                    {
                        flag = false;
                        break;
                    }
            return flag;
        }

    public static void main(String[] args)
     {
         System.out.println("\n在1~200范围内，现在开始证实哥德巴赫猜想：");
         if (Testify_Guess(1, 200))
         {
             System.out.println("\n在 1~200范围内，哥德巴赫猜想是正确的。");
         }
         else
         {
             System.out.println("\n哥德巴赫猜想是错误的");
         }
     }
}
```

程序输出结果如图 14-4 所示。

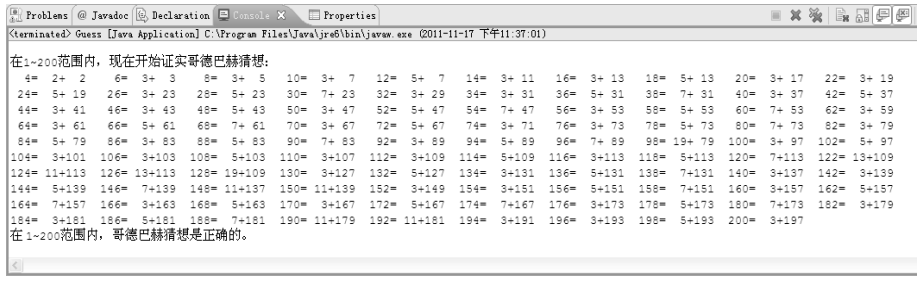

图 14-4　程序输出结果

14.1.3　将一个正整数分解质因数

考题题干：

例如，输入 90，打印出 90=2×3×3×5

考题分析：

对 *n* 进行分解质因数，应先找到一个最小的质数 *k*，然后按下述步骤完成：

（1）如果这个质数恰等于 *n*，则说明分解质因数的过程已经结束，打印出即可。

（2）如果 *n* 不等于 *k*，但 *n* 能被 *k* 整除，则应打印出 *k* 的值，并用 *n* 除以 *k* 的商，作为新的正整数，重复执行第一步。

（3）如果 *n* 不能被 *k* 整除，则用 *k*+1 作为 *k* 的值，重复执行第一步。

参考代码：

```
import java.util.Scanner;

public class fenjie2
{
    int[] result=new int[100];
    int x=0;

    public static boolean isPrime(int n)
    {
        if(n==2)
            return true;
        for(int i=2;i<Math.sqrt(n);i++)
        {
            if(n%i==0)
                return false;
        }
        return true;
    }

    public static int[] minPrime(int n)
    {
        int[] pri=new int[100];
        int a=0;
        for(int i=2;i<=n;i++)
        {
            if(isPrime(i))
            {
                pri[a++]=i;
            }
```

```
        }
        return pri;
}

    public void dispose(int n)
    {
        int j;
        int[] min=fenjie2.minPrime(n);
        for(j=0;j<min.length;j++)
        {
            if(n%min[j]==0)
            {
                this.result[x++]=min[j];
                if(n==min[j]) return ;
                break;
            }
        }
        this.dispose(n/min[j]);
    }

    public static void main(String[] args) throws Exception
    {
        Scanner input=new Scanner(System.in);

        fenjie2 obj=new fenjie2();
        String s="";
        String go;

        System.out.println("请输入数字: ");
        int num=input.nextInt();
        obj.dispose(num);
        for(int i=0;i<obj.result.length;i++)
        {
            if(obj.result[i]>0)
            {
                s +=obj.result[i]+"*";
            }

        }
        System.out.println(num+"="+s.substring(0,s.length()-1));

        System.out.println("结束! ");

    }
}
```

程序输出结果如图 14-5 所示 。

图 14-5　程序输出结果

14.1.4　怎样实现金额转换

考题题干：

金额转换，阿拉伯数字的金额转换成汉字大写的形式如下。

（￥1011）→（壹仟零壹拾壹元整）输出。

考题分析：

金额转换，在开发财务相关软件时会经常用到，也是软件本地化的一个需要。一般开发公司或者团队都有相应的金额转换类或者模块，配合报表工具，可以实现良好的本地化。下面给出一个简单的金额转换代码，供读者参考。

总体思路：对数字进行分级处理，级长为 4，对分级后的每级分别处理，处理后得到字符串相连，例如，123456=12|3456，第二级：12=壹拾贰 + “万”，第一级：3456 =叁仟肆佰伍拾陆 + “”

参考代码如下：

```
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.Scanner;

public final class RMB
{
    private double amount = 0.0D;
    private static final String NUM = "零壹贰叁肆伍陆柒捌玖";
    private static final String UNIT = "仟佰拾个";
    private static final String GRADEUNIT = "仟万亿兆";
    private static final String DOTUNIT = "角分厘";
    private static final int GRADE = 4;
    private static final String SIGN = "￥";
    private static final NumberFormat nf = new DecimalFormat("#0.###");

    private RMB(double amount)
    {
       this.amount = amount;

    }

    public static String toBigAmt(double amount)
    {
        nf.setMinimumFractionDigits(3);     //小数点后不足的补零
        String amt = nf.format(amount);     //将double类型的数格式化并转换成字符串，
        实际上进行了四舍五入
        System.out.println(amt);
        Double d = new Double(amount);
        String dotPart = "";                //取小数位
        String intPart = "";                //取整数位

        int dotPos;
        if ((dotPos = amt.indexOf('.')) != -1)
        {
            intPart = amt.substring(0, dotPos);
            dotPart = amt.substring(dotPos + 1);

            if(dotPart.length()>4)
```

```
            { //超过4位直接截取
                dotPart = dotPart.substring(0,4);
            }
        }
        else
        {
            intPart = amt;
        }

        if (intPart.length() > 16)
            throw new java.lang.InternalError("数额太大，无法转换。");

        String intBig = intToBig(intPart);
        String dotBig = dotToBig(dotPart);

        //以下代码稍做修改，现在好多了。

        if ((dotBig.length() == 0) && (intBig.length() != 0))
        {
            return intBig + "元整";
        }
        else if ((dotBig.length() == 0) && (intBig.length() == 0))
        {
            return intBig + "零元";
        }
        else if ((dotBig.length() != 0) && (intBig.length() != 0))
        {
            return intBig + "元" + dotBig;
         }
        else
        {
            return dotBig;
        }
    }

    /**

     * 用来处理几角几分几厘

     * @param dotPart

     * @return

     */

    private static String dotToBig(String dotPart)
    {
        //得到转换后的大写（小数部分）
        String strRet = "";
        for (int i = 0; i < dotPart.length() && i < 3; i++)
        {
            int num;
            if ((num = Integer.parseInt(dotPart.substring(i, i + 1))) != 0)
                strRet += NUM.substring(num, num + 1)
                        + DOTUNIT.substring(i, i + 1);
        }
```

```
    return strRet;
}

/**
 * 用来处理多少元 , 要仔细考虑才行
 * @param intPart
 * @return
 */
private static String intToBig(String intPart)
{
    //得到转换后的大写（整数部分）
    int grade;                                          //级长
    String result = "";
    String strTmp = "";
            //得到当前级长
    grade = intPart.length() / GRADE;

            //调整级次长度
    if (intPart.length() % GRADE != 0)
        grade += 1;
            //对每级数字处理，先处理最高级的，然后再处理低级的
    for (int i = grade; i >= 1; i--)
    {
        strTmp = getNowGradeVal(intPart, i);        //取得当前级次数字
        result += getSubUnit(strTmp);               //转换大写
        result = dropZero(result);                  //除零外去掉连续的零

            //加级次单位
        if (i > 1)                                  //末位不加单位
            //单位不能相连续
            //注意: 连续4个零的时候要特殊处理(wmj修改此bug)
           if(getSubUnit(strTmp).equals("零零零零"))
           {
              result = result+"零";
            }
           else
           {
              result += GRADEUNIT.substring(i - 1, i);
           }

    }
    return result;
}

private static String getNowGradeVal(String strVal, int grade)
{
            //得到当前级次的串
    String rst;
    if (strVal.length() <= grade * GRADE)
        rst = strVal.substring(0, strVal.length() - (grade - 1) * GRADE);
    else
        rst = strVal.substring(strVal.length() - grade * GRADE, strVal
              .length()
              - (grade - 1) * GRADE);
```

```
    return rst;
}

private static String getSubUnit(String strVal)
{
                //数值转换
    String rst = "";

    for (int i = 0; i < strVal.length(); i++)
    {
        String s = strVal.substring(i, i + 1);
        int num = Integer.parseInt(s);

        if (num == 0)
        {
            //“零”作特殊处理
            if (i != strVal.length())            //转换后数末位不能为零
                rst += "零";
         }
        else
        {
            //If IntKey = 1 And i = 2 Then
            //“壹拾”作特殊处理
            //“壹拾”合理
            rst += NUM.substring(num, num + 1);
            //追加单位

            if (i != strVal.length() - 1)        //个位不加单位
                rst += UNIT.substring(i + 4 - strVal.length(), i + 4

                      - strVal.length() + 1);
        }

    }
    return rst;
}

private static String dropZero(String strVal)
{
    //去除连继的“零”
    String strRst;
    String strBefore;                            //前一位置字符
    String strNow;                               //现在位置字符

   strBefore = strVal.substring(0, 1);
   strRst = strBefore;

   for (int i = 1; i < strVal.length(); i++)
    {
        strNow = strVal.substring(i, i + 1);
        if (strNow.equals("零") && strBefore.equals("零"))
          ;//同时为零
         else
            strRst += strNow;
```

```
            strBefore = strNow;
         }

        //末位去零
        if (strRst.substring(strRst.length() - 1, strRst.length()).equals("零"))
            strRst = strRst.substring(0, strRst.length() - 1);
        return strRst;
    }
    public static void main(String[] args)
    {
    String  rmb;
    double  str;
    System.out.println("请输入金额数: ");
        Scanner scanner = new Scanner(System.in);
        str = scanner.nextDouble();
        System.out.println("转换后的金额为: ");

        rmb=RMB.toBigAmt(str);
        System.out.println(rmb);
    }
}
```

程序输出结果如图 14-6 所示。

图 14-6　程序输出结果

14.1.5　数字排列

考题题干：

用 1、2、2、3、4、5 这 6 个数字，写一个方法，打印出所有不同的排列，如 512234、412345 等，要求 4 不能在第三位，3 与 5 不能相连。

考题分析：

（1）把问题归结为图结构的遍历问题。实际上 6 个数字就是 6 个结点，把 6 个结点连接成无向连通图，对于每一个结点求这个图形的遍历路径，所有结点的遍历路径就是最后对这 6 个数字的排列组合结果集。

（2）显然这个结果集还未达到题目的要求。从以下几个方面考虑。

- 3 与 5 不能相连：实际要求这个连通图的结点 3 与 5 之间不能连通，可在构造图结构时就满足该条件，然后再遍历图。
- 不能有重复：考虑到有两个 2，明显会存在重复结果，可以把结果集放在 TreeSet 中过滤重复结果。
- 4 不能在第三位：仍旧在结果集中去除满足 4 在第三位的结果。

参考代码如下：

```
import java.util.Iterator;
import java.util.TreeSet;

public class shuzi1
{
    private String[] b = new String[]{"1", "2", "2", "3", "4", "5"};
    private int n = b.length;
    private boolean[] visited = new boolean[n];

    private int[][] a = new int[n][n];
    private String result = "";
    private TreeSet TreeSet = new TreeSet();

    public static void main(String[] args)
    {
        new shuzi1().start();
    }

    private void start()
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (i == j)
                {
                    a[i][j] = 0;
                }
                else
                {
                    a[i][j] = 1;
                }
            }
        }
        a[3][5] = 0;
        a[5][3] = 0;
        for (int i = 0; i < n; i++)
        {
            this.depthFirstSearch(i);
        }
        Iterator it = TreeSet.iterator();
        while (it.hasNext())
        {
            String string = (String) it.next();

            if (string.indexOf("4") != 2)
            {
                System.out.println(string);
            }
        }
    }

    private void depthFirstSearch(int startIndex)
```

```
    {
        visited[startIndex] = true;
        result = result + b[startIndex];
        if (result.length() == n)
        {
            TreeSet .add(result);
        }
        for(int j = 0; j < n; j++)
        {
            if (a[startIndex][j] == 1 && visited[j] == false)
            {
                depthFirstSearch(j);
            }
            else
            {
                continue;
            }
        }
        result = result.substring(0, result.length() -1);
        visited[startIndex] = false;
    }
}
```

程序输出结果如图 14-7 所示。

图 14-7　程序输出结果

14.1.6　数字拆解

考题题干：

将任一个数字进行拆解，最大数不超过 2，例如。

3 = 2+1=1+2 = 1+1+1，　共 3 种拆法。

4 = 3 + 1 = 2 + 2 = 2 + 1 + 1 =1+2+1= 1 + 1 + 1 + 1，　共 5 种拆法。

5=2+2+1=2+1+2=2+1+1+1=1+2+2=1+2+1+1=1+1+2+1=1+1+1+2=1+1+1+1+1，共 8 种拆法。

随便给一个数字，对其进行拆解，并打印可拆解情况和拆解结果数。

考题分析：

假设已成功拆出有加数为 *m* 个的数，现在考虑拆出 *m*+1 个的情况，即可以看成把已经成功拆出的 *m* 个中的某一个拆成两个，且如果这 *m* 个中存在至少两个相等，只需拆出其中一个为两个就行了。直到最后的 *n* 个加数。对于把一个数拆成两个数的方式，这很简单。此方案也应该是可行的。

可以用一个递归来实现，对于任意的整数 n，记拆解方式为 g，且做以下假设：

- 最后一个拆数是 1，则拆解方式为 $g(n-1)+1$，这个+1 表示在 g(n－1)的末尾加上 1。
- 最后一个拆数是 2，则拆解方式为 $g(n-2)+2$，依此类推，直到最后一个拆解数为 n－1。

参考代码如下：

```
import java.io.*;
import java.util.Scanner;

class upstair
{
   static int sum(int a[])
   {
       int sum=0;
        for(int i=1;i<a.length;i++)
            sum+=a[i];
       return sum;
    }

    static void up(int n)
{
       int a[]=new int[n+1];
        int count=0;
        int i,k;
        a[0]=-2;
        for(i=1;i<n+1;i++)
            a[i]=1;
        k=n;
        while(a[k]!=-1){
            if(sum(a)<n)  k++;
            if(sum(a)==n){
              for(i=1;i<=k;i++)
                  System.out.print(a[i]+" ");
                System.out.println();
                count++;
              a[k]=0;
              while(a[--k]==2)
                   a[k]=0;
            }
            a[k]++;
       }
       System.out.println(n+"的拆分方法共有:"+count+"种。");
    }

public static void main(String argc[])
{
        int n;
       Scanner input=new Scanner(System.in);
        System.out.println("请输入数字: ");
        n=input.nextInt();
        up(n);
     }
}
```

程序输出结果如图 14-8 所示。

图 14-8　程序输出结果

14.1.7　数字组合

考题题干：

将 1、2、3、4、5、6、7、8、9 这 9 个数字分成三个百位数，每个数字用且只用一次，并且第三个数是第一个数的 3 倍，第二个数是第一个数的两倍。求三个数。说明，结果可能多于一组，如 327、654、981。

考题分析：

第一个数百位数最大为 3，可谓一层循环，十位和个位则组成两层循环。用一个 Set 容器来存储第一个三位数，利用第一个数求出第三个数（是第一个数的 3 倍），把第三个数的每一位求出，看是否符合，如符合则加入 Set 容器，而这时只剩下三个数，对这三个数的 6 种情况求解，等于第一个数的两倍即成立。

参考代码如下：

```
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class test39
{
    public static void main(String[] args)
    {
        int i,j,k,m,n,t1,t2,t3;
        Set total=new HashSet();
        for(int t=1;t<10;t++)
        {
            total.add(t);
        }
        Set set=new HashSet();
        Set lastThree=new HashSet();
        for(i=1;i<4;i++)
        {
            set.add(i);
            for(j=1;j<10;j++)
            {
                if(j==i)
                {
                    continue;
```

```
}
set.add(j);
for(k=1;k<10;k++)
{
    if(k==i||k==j)
    {
        continue;
    }
    set.add(k);
    m=3*(i*100+j*10+k);
    if(m>987)
    {
        set.remove(k);continue;
    }//如果已超出范围，没必要再往下判断
    t1=m%10; if(t1==0)
    {
        set.remove(k);
        continue;
    } //个位数
    t2=(m%100)/10;
    if(t2==0)
    {
        set.remove(k);
        continue;
    } //十位数
    t3=m/100;
    if(t3==0)
    {
        set.remove(k);
        continue;
    } //百位数
    if(set.contains(t1))
    {
        set.remove(k);
        continue;
    }
    set.add(t1);
    if(set.contains(t2))
    {
        set.remove(k);
        set.remove(t1);
        continue;
    }
    set.add(t2);
    if(set.contains(t3))
    {
        set.remove(k);
        set.remove(t1);
        set.remove(t2);
        continue;
    }
    set.add(t3);

    total.removeAll(total);
```

```
                for(int q=1;q<10;q++)
                {
                    total.add(q);
                }
                total.removeAll(set);
                int[] s=new int[3];
                Iterator iterator=total.iterator();
                int p=0;
                while(iterator.hasNext())
                {
                    s[p]=Integer.parseInt(iterator.next()+"");
                    p++;
                }
                n=s[0]*100+s[1]*10+s[2];
                if(n==2*(i*100+j*10+k))
                {
                    System.out.println(i+""+j+""+k+"    "+n+"    "+m);
                }
                n=s[0]*100+s[2]*10+s[1];
                if(n==2*(i*100+j*10+k))
                {
                    System.out.println(i+""+j+""+k+"    "+n+"    "+m);
                }
                n=s[1]*100+s[0]*10+s[2];
                if(n==2*(i*100+j*10+k))
                {
                    System.out.println(i+""+j+""+k+"    "+n+"    "+m);
                }
                n=s[1]*100+s[2]*10+s[0];
                if(n==2*(i*100+j*10+k))
                {
                    System.out.println(i+""+j+""+k+"    "+n+"    "+m);
                }
                n=s[2]*100+s[0]*10+s[1];
                if(n==2*(i*100+j*10+k))
                {
                    System.out.println(i+""+j+""+k+"    "+n+"    "+m);
                }
                n=s[2]*100+s[1]*10+s[0];
                if(n==2*(i*100+j*10+k))
                {
                    System.out.println(i+""+j+""+k+"    "+n+"    "+m);
                }
                set.remove(t1);
                set.remove(t2);
                set.remove(t3);
                set.remove(k);
            }
            set.remove(j);
        }
        set.remove(i);
    }
  }
}
```

程序输出结果如图 14-9 所示。

图 14-9　程序输出结果

14.2　思维扩展算法

思维扩展算法是指根据题目的描述，找到解决问题的计算模型，然后设计算法。这类题目的关键是知道正确的编程模型。主要包括：蛇形打印、24 点游戏、掷骰子等。

14.2.1　蛇形打印

考题题干：

打印出蛇形图案，如图 14-10 所示。

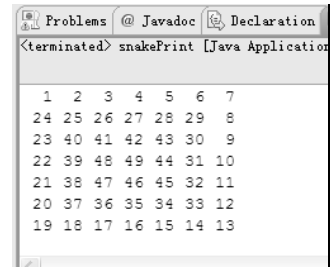

图 14-10　蛇形图案

考题分析：

根据图案效果，可以考虑用二维数组，控制输出的关键在于方向变化的地方，行列长度结合初始化的值变化，由外而内，逐个赋值，最后循环输出二维数组。

参考代码如下：

```
class snakePrint
{
    static int length = 8;
    static int value = 1;
    static int[][] snake = new int[length][length];
    static Direction lastDirection = Direction.Right;

static enum Direction
{
        Right, Down, Left, Up;
    }

public static void initialArray()
{
        int row = 0, col = 0;
        for (int c = 0; c < length * length; c++) {
            snake[row][col] = value;
            lastDirection = findDirection(row, col);
            switch (lastDirection) {
                case Right:
                    col++;
                    break;
```

```
                case Down:
                    row++;
                    break;
                case Left:
                    col--;
                    break;
                case Up:
                    row--;
                    break;
                default:
                    System.out.println("error");
            }
            value++;
        }
    }

static Direction findDirection(int row, int col)
{
        Direction direction = lastDirection;
        switch (direction) {
            case Right: {
                if ((col == length - 1) || (snake[row][col + 1] != 0))
                    direction = direction.Down;
                break;
            }
            case Down: {
                if ((row == length - 1) || (snake[row + 1][col] != 0))
                    direction = direction.Left;
                break;
            }
            case Left: {
                if ((col == 0) || (snake[row][col - 1] != 0))
                    direction = direction.Up;
                break;
            }
            case Up: {
                if (snake[row - 1][col] != 0)
                    direction = direction.Right;
                break;
            }
        }
        return direction;
    }

public static void main(String[] args)
{
        initialArray();
        for (int i = 0; i < length; i++)
        {
            for (int j = 0; j < length; j++)
            {
              System.out.printf(" %2d",snake[i][j]);
```

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

程序输出结果如图 14-11 所示。

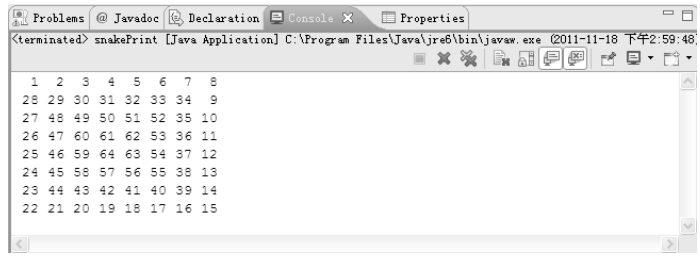

图 14-11　程序输出结果

14.2.2　24 点算法

考题题干：

随机给 4 个数字，对其进行加减乘除四则运算，最终值为 24，并打印出数学表达式。

考题分析：

基本原理是穷举 4 个整数所有可能的表达式，然后对表达式求值。

表达式的定义：expression = (expression|number) operator (expression|number)。

因为能使用的 4 种运算符 + - * / 都是二元运算符，所以这里只考虑二元运算符。二元运算符接收两个参数，输出计算结果，并参与后续的计算。

构造所有可能的表达式的算法如下：

（1）将 4 个整数放入数组中。

（2）在数组中取两个数字的排列，共有 P（4,2）种排列。对每一个排列。

（2-1）用 + - * / 每一个运算符，

（2-1-1）根据此排列的两个数字和运算符计算结果。

（2-1-2）改表数组：将此排列的两个数字从数组中去除，将步骤 2-1-1 计算的结果放入数组中。

（2-1-3）对新的数组重复步骤（2）。

（2-1-4）恢复数组：将此排列的两个数字加入数组中，将步骤 2-1-1 计算的结果从数组中去除。

可见这是一个递归过程。步骤（2）是递归函数。当数组中只剩下一个数字时，就是表达式的最终结果，此时递归结束。

在程序中，一定要注意递归的现场保护和恢复，即递归调用之前与之后，现场状态应该保持一致。在上述算法中，递归现场是指数组，步骤 2-1-2 改变数组以进行下一层递归调用，步骤 2-1-3 则恢复数组，以确保当前递归调用获得下一个正确的排列。

括号 () 的作用只是改变运算符的优先级，即运算符的计算顺序。所以在上述算法中，无须考虑括号。括号在输出时需加以考虑。

参考代码如下：

```
import java.util.*;

class Expression
```

```
{
    private ArrayList<Integer> num=new ArrayList<Integer>();
    private ArrayList<String> sign=new ArrayList<String>();

    public void add(int num2)
    {
       num.add(num2);
    }
    public void add(String sign2)
    {
       sign.add(sign2);
    }
    private int getPriority(String sign)
    {
       if (sign.equals("+")) return 1;
       if (sign.equals("-")) return 1;
       if (sign.equals("*")) return 2;
       if (sign.equals("/")) return 2;
       return -1;
    }

    private String toString(int layer)
    {
       if (layer==0)
       {
          return num.get(0)+sign.get(0)+num.get(1);
       }
       else
       {
         String result=this.toString(layer-1);
         if (getPriority(sign.get(layer))>getPriority(sign.get(layer-1)))
            result="("+result+")";
         result+=sign.get(layer)+num.get(layer+1);
         return result;
       }
    }

    public String toString()
    {
       return toString(2);
    }

    public void clear()
    {
       num.clear();
       sign.clear();
    }
}

class Thinker
{
    private Expression exp;

    public Thinker(Expression expression)
```

```
{
    exp=expression;
}

public boolean count(ArrayList<Integer> array,int num,int target)
{
    if (num==1)
    {
        if ((array.get(0)+array.get(1))==target)
        {
            exp.add(array.get(0));
            exp.add(array.get(1));
            exp.add("+");
            return true;
        }
        if ((array.get(0)-array.get(1))==target)
        {
            exp.add(array.get(0));
            exp.add(array.get(1));
            exp.add("-");
            return true;
        }
        if ((array.get(1)-array.get(0))==target)
        {
            exp.add(array.get(1));
            exp.add(array.get(0));
            exp.add("-");
            return true;
        }
        if ((array.get(0)*array.get(1))==target)
        {
            exp.add(array.get(0));
            exp.add(array.get(1));
            exp.add("*");
            return true;
        }
        if ((array.get(0)*target==array.get(1)))
        {
            exp.add(array.get(1));
            exp.add(array.get(0));
            exp.add("/");
            return true;
        }
        if ((array.get(1)*target==array.get(0)))
        {
            exp.add(array.get(0));
            exp.add(array.get(1));
            exp.add("/");
            return true;
        }
        return false;
    }
    else
    {
```

```
for (int current=0;current<array.size();current++)
{
   ArrayList<Integer> array1=new ArrayList<Integer>();
   int currentNum=array.get(current);
   for (int i=0;i<array.size();i++)
   if (i!=current) array1.add(array.get(i));
   if (count(array1,num-1,target-currentNum))
   {
      exp.add("+");
      exp.add(currentNum);
      if (num==3)
      {
         System.out.println(exp.toString());
         exp.clear();
      }
      if (num!=3) return true;
   }
   if (count(array1,num-1,target+currentNum))
   {
      exp.add("-");
      exp.add(currentNum);
      if (num==3)
      {
         System.out.println(exp.toString());
         exp.clear();
      }
      if (num!=3) return true;
   }
   if (count(array1,num-1,target*currentNum))
   {
      exp.add("/");
      exp.add(currentNum);
      if (num==3)
      {
         System.out.println(exp.toString());
         exp.clear();
      }
      if (num!=3) return true;
   }
   if (target%currentNum==0)
   {
      if (count(array1,num-1,(int)(target/currentNum)))
      {
         exp.add("*");
         exp.add(currentNum);
         if (num==3)
            {
            System.out.println(exp.toString());
            exp.clear();
         }
         if (num!=3) return true;
      }
   }
}
```

```
            return false;
        }
    }
}

public class ersi2
{
   public static void main(String[] args)
{
       // TODO Auto-generated method stub
       Expression expression=new Expression();
       Thinker thinker=new Thinker(expression);
       ArrayList<Integer> card=new ArrayList<Integer>();
       int target=24;

       Scanner input=new Scanner(System.in);
       System.out.println("请输入4个数字: ");
       int t;
       for(int i=0;i<4;i++)
       {
          t=input.nextInt();
          card.add(t);
       }
       thinker.count(card,card.size()-1,target);
   }
}
```

程序输出结果如图 14-12 所示。

图 14-12　程序输出结果

14.2.3　双色球随机摇号

考题题干：

用计算机模拟双色球摇号，根据福利彩票双色球玩法规则，6 个蓝色球，数字范围 1~32，不允许重复，1 个红色球，范围 1~16，用计算机自动生成 6 个蓝色球，1 个红色球。

考题分析：

双色球自动生成 7 个号码，蓝色球和红色球分别生成，其中 6 个蓝色球要求不允许重复，可以考虑直接用 set 来实现，还可以把不重复的数字先存放到数组中，然后循环从数组中取数，取一次调整一下数组，实现每次取值不重复。

参考代码如下：

```
import java.util.Arrays;
```

```
import java.util.Random;
import java.util.Scanner;

public class caipiao1
{
    /**
     * 根据给定的最小数字和最大数字，以及随机数的个数，产生指定的不重复的数组
     * @param begin 最小数字（包含该数）
     * @param end 最大数字（不包含该数）
     * @param size 指定产生随机数的个数
     */
    public static int[] generateRandomNumber(int begin, int end, int size) {
        // 加入逻辑判断，确保begin<end并且size不能大于该表示范围
        if (begin >= end || (end - begin) < size)
        {
            return null;
        }
        //可以随意生成，但不能重复
        int[] seed = new int[end - begin];

        for (int i = begin; i < end; i ++)
        {
            seed[i - begin] = i;
        }
        int[] ranArr = new int[size];
        Random ran = new Random();
        // 数量可以自己定义
        for (int i = 0; i < size; i++)
        {
            // 得到一个位置
            int j = ran.nextInt(seed.length - i);
            // 得到那个位置的数值
            ranArr[i] = seed[j];
            // 将最后一个未用的数字放到这里
            seed[j] = seed[seed.length - 1 - i];
        }
        return ranArr;
    }

    public static void main(String[] args)
    {
        int[] ranArr={};
        int red;
        Scanner input=new Scanner(System.in);
        Random ran = new Random();

        System.out.println("欢迎使用双色球自动摇号系统");
        System.out.print("确实摇号(y/n)?");
        String go;
        go=input.next();

        while(go.equalsIgnoreCase("y")){
```

```
            ranArr= generateRandomNumber(1,33,6);
            red=ran.nextInt(16);
            System.out.println(Arrays.toString(ranArr)+" "+red);
            System.out.print("继续摇号(y/n)?");
            go=input.next();
        }
        System.out.println("谢谢使用! ");
    }

}

import java.util.Arrays;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Random;
import java.util.Scanner;
import java.util.Set;

public class caipiao2
{
    /**
     * 根据给定的最小数字和最大数字，以及随机数的个数，产生指定的不重复的数组
     *
     * @param begin
     *          最小数字（包含该数）
     * @param end
     *          最大数字（不包含该数）
     * @param size
     *          指定产生随机数的个数
     */
    public static int[] generateBySet1(int begin, int end, int size)
    {
        // 加入逻辑判断，确保begin<end并且size不能大于该表示范围
        if (begin >= end || (end - begin) < size)
        {
            return null;
        }

        Random ran = new Random();
        Set<Integer> set = new HashSet<Integer>();
        while (set.size() < size)
        {
            set.add(begin + ran.nextInt(end - begin));
        }

        int[] ranArr = new int[size];
        Iterator it = set.iterator();

        for (int i = 0; i < size; i ++)
        {
            if (!it.hasNext())
            {
                break;
```

```
            }
            ranArr[i] = (Integer) it.next();
        }
        return ranArr;
    }

    public static void main(String[] args)
    {
        int[] ranArr={};
            int red;
            Scanner input=new Scanner(System.in);
        Random ran = new Random();

        System.out.println("欢迎使用双色球自动摇号系统");
        System.out.print("确实摇号(y/n)?");
        String go;
        go=input.next();

        while(go.equalsIgnoreCase("y"))
        {
                ranArr= generateBySet1(1,33,6);
                red=ran.nextInt(16);
             System.out.println(Arrays.toString(ranArr)+" "+red);
                System.out.print("继续摇号(y/n)?");
                go=input.next();
        }
        System.out.println("谢谢使用！");
}
}
```

程序输出结果如图 14-13 所示。

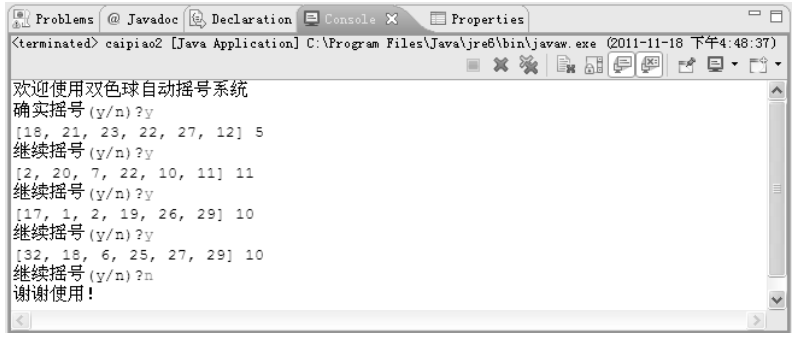

图 14-13　程序输出结果

14.2.4　巧妙过桥

考题题干：

小明一家过一座桥，过桥的时候是黑夜，所以必须有灯。现在小明过桥要 2 分钟，小明的弟弟要 5 分钟，小明的爸爸要 6 分钟，小明的妈妈要 9 分钟，小明的爷爷要 13 分钟。每次此桥最多可以过两人，而过桥的速度根据过桥最慢者而定，而且灯在点燃后 40 分钟就会熄灭。问小明一家如何过桥时间最短？

试题分析：

下面用程序来求解。

参考代码如下：

```
public class bridge
{
    static int index;                                    //过桥临时方案的数组下标
    static int size = 64;
    static int N = 5;
    static int mintime = 40;                             //最小过桥时间总和，初始值30
    static int[] transit = new int[size];                //进行下标中转的数组
    static int program[] = new int[size];                //最短时间内过桥的方案
    static int time[] = {2, 5, 6, 9, 13};                //每个人过桥所需要的时间
    /*
     * 将人员编号：小明 location[0]，弟弟location[1]，
     * 爸爸location[2]，妈妈location[3]，爷爷location[4] 每个人的当前位置：0--在桥左边，
     1--在桥右边
     */
    static int location[] = new int[N];

    /*
     * 参数说明：notPass:未过桥人数;usedtime:当前已用时间;Direction:过桥方向,1--向右，
     0--向左
     */
    public static void Find(int notPass, int usedtime, int Direction)
    {
        if (notPass == 0)
        {                                    //所有人已经过桥，更新最少时间及方案
            mintime = usedtime;
            for (int i = 0; i < size && transit[i] >= 0; i++)
            {
                program[i] = transit[i];
            }
        }
        else if (Direction == 1)
        {                                    //过桥方向向右，从桥左侧选出两人过桥
            for (int i = 0; i < N; i++)
            {
                if (location[i] == 0 && (usedtime + time[i]) < mintime)
                {
                    transit[index++] = i;
                    location[i] = 1;
                    for (int j = 0; j < N; j++)
                    {
                        int TmpMax = (time[i] > time[j] ? time[i] : time[j]);
                        if (location[j] == 0 && (usedtime + TmpMax) < mintime)
                        {
                            transit[index++] = j;
                            location[j] = 1;
                            Find((notPass - 2), (usedtime + TmpMax), 0);
                            location[j] = 0;
                            transit[--index] = -1;
                        }
                    }
```

```
                location[i] = 0;
                transit[--index] = -1;
            }
        }
    }
    else
    {                                   //过桥方向向左，从桥右侧选出一个人回来送灯
        for (int j = 0; j < N; j++)
        {
            if (location[j] == 1 && usedtime + time[j] < mintime)
            {
                transit[index++] = j;
                location[j] = 0;
                Find(notPass + 1, usedtime + time[j], 1);
                location[j] = 1;
                transit[--index] = -1;
            }
        }
    }
}

public static void main(String[] args)
{
String s1="",s2="",s3="";
    for (int i = 0; i < size; i++)
    {
        program[i] = -1;
        transit[i] = -1;              //初始方案内容为负值，避免和人员标号冲突
    }
    Find(N, 0, 1);                                      //查找最佳方案
    System.out.println("最短过桥时间为: " + mintime); //输出最短过桥时间
    System.out.println("最佳过桥组合为: ");            //输出最佳过桥组合
    for (int i = 0; i < size && program[i] >= 0; i += 3)
    {
        switch (program[i])
        {
            case 0:
                s1="小明";
                break;
            case 1:
                s1="弟弟";
                break;
            case 2:
                s1="爸爸";
                break;
            case 3:
                s1="妈妈";
                break;
            case 4:
                s1="爷爷";
                break;
        }
          switch (program[i+1])
        {
```

```
                case 0:
                    s2="小明";
                    break;
                case 1:
                    s2="弟弟";
                    break;
                case 2:
                    s2="爸爸";
                    break;
                case 3:
                    s2="妈妈";
                    break;
                case 4:
                    s2="爷爷";
                    break;
            }
              switch (program[i+2])
            {
                case 0:
                    s3="小明";
                    break;
                case 1:
                    s3="弟弟";
                    break;
                case 2:
                    s3="爸爸";
                    break;
                case 3:
                    s3="妈妈";
                    break;
                case 4:
                    s3="爷爷";
                    break;
                case -1:
                    s3="没人";
                    break;
            }

            System.out.println(s1 + "和" + s2+"一道过桥，"+ s3+"再回来");
        }
    }

}
```

程序输出结果如图 14-14 所示。

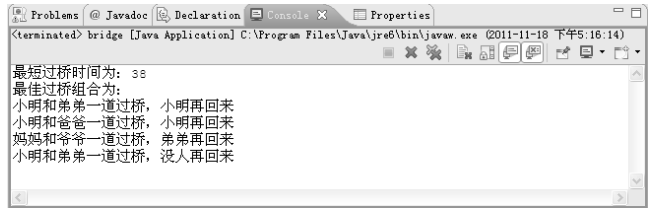

图 14-14　程序输出结果

14.2.5　猴子吃桃

考题题干：

猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个；第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第 10 天早上想再吃时，见只剩下一个桃子了。求第一天共摘了多少。

考题分析：

采取逆向思维的方法，从后往前推断。

参考代码：

```
public class monkey
{
    static int total(int day)
{
        if(day == 10)
        {
        return 1;
        }
        Else
        {
        return (total(day+1)+1)*2;
        }
    }

public static void main(String[] args)
{
       System.out.println(total(1));
}
}
```

程序输出结果如图 14-15 所示（注：本题前面章节中有过讲解，但此处的算法更简洁，读者可比较一下）。

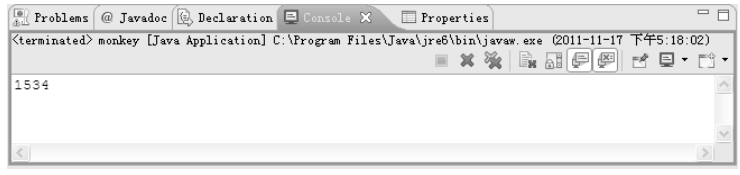

图 14-15　程序输出结果

14.2.6　天平称物

考题题干：

有 4 个砝码，总重量为 40 克，砝码的重量是整数，且各不相等。请确定它们的重量，使之能称出 1～40 克之间任何整数重量的物体。

考题分析：

（1）一个砝码重量生成器。生成 4 个砝码各个重量的所有组合。

（2）一个验证程序。验证此砝码组合是否符合标准。

生成从 1 到 40 克的物体重量，每一种物体重量验证一次是否可行。砝码的使用方法只有

如下两种：

- 物体重量=砝码重量；
- 砝码重量[左]+物体重量=砝码重量[右]。

第一种情况的所有组合：从 4 个砝码里面任意取出 1 个砝码，两个、3 个，或者 4 个。

依次套用第一种情况的公式，有一次成功则符合，否则看第二种情况；。

第二种情况的所有组合：从 4 个砝码里面任意取出 1 个砝码，两个、3 个，或者 4 个，总重量赋给砝码重量[左]；从剩下的 4 个砝码里面任意取出 1 个砝码，2 个、3 个（小于剩余砝码总数），赋给砝码重量[右]，依次套用第二种情况的公式，依次成功则符合，否则不符合，换下一个砝码重量组合。

（3）显示通过验证程序的砝码组合。

参考代码如下：

```
import  java.util.ArrayList;

public  class  fama2  {
       private  int  count;
       private  ArrayList  A=new  ArrayList();
       private  ArrayList  B=new  ArrayList();
       fama2(int  count)
       {
           if  (count  >  0)
           {
              this.count  =  count;
           }
       }
       public  void  prtGroup()
       {
             int  Left,Inserted,temp;
             Left=count;
             Inserted=0;
             temp=Inserted*2+1;
             int  index=0;
             int  i;
             while(Left> 0)
             {
                  temp=Math.min(temp,Left);
                  A.add(index,new  Integer(temp));
                  for(i=Inserted+1;i <=Inserted+temp;i++)
                  {
                      if(i <temp)
                      {
                          B.add(i-1,new  String(temp+ "-( "+B.get(temp-i-1)+ ") "));
                       }
                       else  if(i==temp)
                       {
                          B.add(i-1,Integer.toString(temp));
                        }
                        else  if(i> temp)
                        {
```

```
                        B.add(i-1,new  String(temp+ "+( "+B.get(i-temp-1)+ ") "));
                    }
                }
                Inserted=Inserted+temp;
                Left=Left-temp;
                temp=Inserted*2+1;
            index++;
        }
        for(i=0;i <A.size();i++)
        {
            System.out.println(A.get(i));
        }
        for(i=0;i <B.size();i++)
        {
           System.out.println((i+1)+ "= "+B.get(i));
        }
    }
    public  static  void  main(String[]  args)
    {
        int  count=40;
        fama2  PS=new  fama2(count);
        PS.prtGroup();
    }
}
```

程序输出结果如图 14-16 所示。

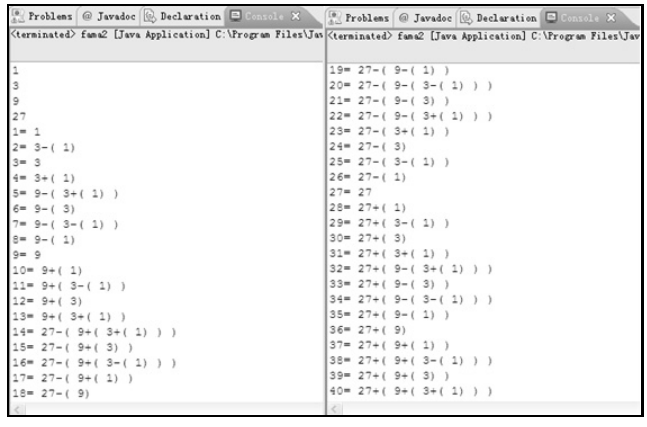

图 14-16　程序输出结果

14.2.7　掷骰子游戏

考题题干：

掷骰子游戏是一个简单的游戏，规则如下。玩家掷两个骰子，点数为 1～6，如果第一次点数和为 7 或 11，则玩家胜；如果点数和为 2、3 或 12，则玩家输；如果和为其他点数，则记录第一次的点数和；然后继续掷骰子，直至点数和等于第一次掷出的点数和，则玩家胜；如果在这之前掷出了点数和为 7，则玩家输。请编程实现。

试题分析：

规则看起来有些复杂，但其实只要使用 switch 配合 if 条件判断来编写代码即可，注意不

要弄错胜负顺序。

参考代码如下：

```
import  java.util.ArrayList;
import  java.util.List;
import  java.util.Random;
Import  java.util.Scanner;

class Craps
{
    public static enum Status
    {
       CONTINUE, WON, LOST
    };

    private List<String> results;
    private int w;
    private int l;

    public Craps()
    {
       results = new ArrayList<String>();
       w = 0;
       l = 0;
    }

    public static Random random = new Random();

    private final static int SNAKE_EYES = 2;
    private final static int TREY = 3;
    private final static int SEVEN = 7;
    private final static int YO_LEVEN = 11;
    private final static int BOX_CARS = 12;

   public void play()
   {
       int myPoint = 0;
       Status gameStatus = Status.CONTINUE;
       int sumOfDice = rollDice();
       switch (sumOfDice)
       {
          case SEVEN:
          case YO_LEVEN:
             gameStatus = Status.WON;
             break;
          case SNAKE_EYES:
          case TREY:
          case BOX_CARS:
             gameStatus = Status.LOST;
             break;
       }

       if(gameStatus == Status.CONTINUE)
```

```
        {
            myPoint = sumOfDice;
            System.out.printf("点数是  %d\n", myPoint);
        }

        while (gameStatus == Status.CONTINUE)
        {
            sumOfDice = rollDice();

            if (sumOfDice == myPoint)
                gameStatus = Status.WON;
            else if (sumOfDice == SEVEN)
                gameStatus = Status.LOST;
        }

        if (gameStatus == Status.WON)
    {
            System.out.println("玩家赢");
            results.add("赢");
            w++;
        }
        else
        {
            System.out.println("玩家输");
            results.add("输");
            l++;
        }
    }

    public int rollDice()
    {
        int die1 = 1 + random.nextInt(6);
        int die2 = 1 + random.nextInt(6);

        int sum = die1 + die2;

        System.out.printf("玩家掷的点数  %d + %d = %d\n", die1, die2, sum);

        return sum;
     }

     public void print() {
        System.out.println("游戏结果统计: ");
        System.out.println("赢的次数:  " + w);
        System.out.println("输的次数:  " + l);
        int i = 0;
        for(String rs: results) {
            System.out.println("第  "+(++i)+"次: " +rs);
        }
     }
}

public class CrapsTest
```

```
{
    public static void main(String args[])
{
     String answer;
        Craps game = new Craps();

        do{
        game.play();

        System.out.println("继续游戏吗(y/n)?");
        Scanner input = new Scanner(System.in);
        answer = input.next();
        }while("Y".equalsIgnoreCase(answer));

        game.print();
    }
}
```

程序结果如图 14-17 所示。

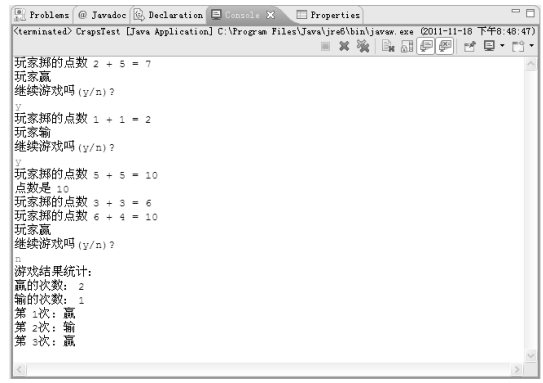

图 14-17　程序输出结果

14.3　小结

算法作为面试的一个主要考查内容，在面试中的地位越来越重要。本章详细讲解了常见的基础算法和思维扩展算法。主要包括字符串匹配算法、数字排列算法、天平称物算法、掷骰子算法等。每一种算法都各具特色，读者可以根据个人喜好，有选择地深入研究，提出自己更好的算法，以达到熟练掌握，为面试奠定坚实基础。