

在线编程题参考答案

第 3 章

3.1 第 1 章 绪论

1. POJ1004——财务管理问题

时间限制：1000ms；空间限制：10 000KB。

问题描述：拉里今年毕业，终于找到了工作。他赚了很多钱，但似乎还没有足够的钱，拉里决定抓住金融投资机会解决他的财务问题。拉里有自己的银行账户报表，他想看看自己有多少钱。请编写一个程序帮助拉里从过去 12 个月的每一月中取出他的期末账户余额并计算他的平均账户余额。

输入格式：输入为 12 行，每行包含特定月份的银行账户的期末余额，每个数字都是正数并到便士为止，不包括美元符号。

输出格式：输出一个数字，即 12 个月的期末账户余额的平均值（平均值），它将四舍五入到最近的便士，紧接着是美元符号，然后是行尾。在输出中不会有其他空格或字符。

输入样例：

```
100.00
489.12
12454.12
1234.10
823.05
109.20
5.27
1542.25
839.18
83.99
1295.01
1.75
```

输出样例:

\$ 1581.42

解: 读取每个月份的期末账户余额 temp, 累加到 sum 中, 输出 sum/12 的结果。对应的 AC 程序如下:

```
import java.util.Scanner;
public class Main
{   public static void main(String[] args)
    {   Scanner fin=new Scanner(System.in);
        double temp=0.00,sum=0.00;
        int iMax=12;
        while(iMax-->0)
        {   temp=fin.nextDouble();
            sum += temp;
        }
        System.out.printf("$ %.2f\n",sum/12);
    }
}
```

上述程序的执行时间为 188ms, 执行空间为 3228KB。

2. HDU2012——素数判定问题

时间限制: 2000ms; 空间限制: 65 536KB。

问题描述: 对于表达式 $n^2 + n + 41$, 当 n 在 (x, y) 范围内取整数值时 (包括 $x, y, -39 \leq x < y \leq 50$) 判定该表达式的值是否都为素数。

输入格式: 输入数据有多组, 每组占一行, 由两个整数 x, y 组成, 当 $x=0, y=0$ 时表示输入结束, 该行不做处理。

输出格式: 对于每个给定范围内的取值, 如果表达式的值都为素数, 输出“OK”, 否则输出“Sorry”, 每组输出占一行。

输入样例:

0 1
0 0

输出样例:

OK

解: 对于整数 num, 若它能够整除 $2 \sim \sqrt{\text{num}}$ 中的任何整数, 则不是素数, 否则为素数。对应的 AC 程序如下:

```
import java.util.Scanner;
public class Main
{   public static void main(String[] args)
    {   Scanner in = new Scanner(System.in);
        while(in.hasNextInt())
        {   int x = in.nextInt();
            int y = in.nextInt();
```

```

        boolean flag = true;
        int number = 0;
        int num = 0;
        if(x==0 && y== 0)
            break;
        else
        {
            if(x>y)
            {
                int temp = x;
                x = y;
                y = temp;
            }
            for(int i = x; i<= y; i++)
            {
                num=i*i+i+41;
                for(int j = 2; j<= Math.sqrt(num); j++)           //素数判断
                {
                    if(num % j == 0)
                        flag = false;
                }
            }
            if(flag == true)
                System.out.println("OK");
            else
                System.out.println("Sorry");
            flag = true;
        }
    }
}
}
}

```

上述程序的执行时间为 296ms,执行空间为 9312KB。

提示：在本书所有在线编程题中,POJ 题目的限制时间和空间是针对 C/C++ 程序的,通常 Java 程序的时间和空间是其两倍,HDU 题目的限制时间和空间就是针对 Java 程序的。

3.2 第2章 线性表

1. HDU2019——数列有序问题

时间限制：2000ms；空间限制：65 536KB。

问题描述：有 n ($n \leq 100$) 个整数,已经按照从小到大的顺序排列好,现在另外给一个整数 m ,请将该数插入序列中,并使新的序列仍然有序。

输入格式：输入数据包含多个测试实例,每组数据由两行组成,第一行是 n 和 m ,第二行是已经有序的 n 个数的数列。 n 和 m 同时为 0 表示输入数据的结束,本行不做处理。

输出格式：对于每个测试实例,输出插入新的元素后的数列。

输入样例：

```

3 3
1 2 4
0 0

```

输出样例:

1 2 3 4

解: 对于每个测试实例, n 个有序整数采用顺序表存放, 为了简单, 这里顺序表直接用数组 a 表示。先查找插入 m 的位置 i , 将 $a[i..n-1]$ 的所有元素均后移一个位置, 置 $a[i]=m$ 。对应的 AC 程序如下:

```
import java.util.*;
public class Main
{
    public static void main(String args[])
    {
        Scanner cin=new Scanner(System.in);
        while(cin.hasNext())                //读取每个测试实例
        {
            int n=cin.nextInt();            //读取 n
            int m=cin.nextInt();            //读取 m
            if(n==0 && m==0)                //结束
                break;
            else
            {
                int a[]=new int[105];        //最多 100 个整数
                for(int i=0;i<n;i++)        //读取递增有序整数序列
                    a[i]=cin.nextInt();
                if(m>a[n-1])                //m 最大, 插入末尾
                    a[n]=m;
                else
                {
                    for(int i=0;i<n;i++)    //查找第一个大于等于 m 的元素 a[i]
                    {
                        if(m<a[i])
                        {
                            int j=i;
                            for( i=n;i>j;i--) //将 a[i..n-1]后移一个位置
                                a[i]=a[i-1];
                            a[i]=m;          //插入 a[i]
                            break;           //退出查找
                        }
                    }
                }
                for(int i=0;i<=n;i++)        //输出新序列
                {
                    if(i==0)
                        System.out.print(a[i]);
                    else
                        System.out.print(" "+a[i]);
                }
                System.out.println();
            }
        }
    }
}
```

上述程序的执行时间为 265ms, 空间为 9412KB。

2. HDU1443——Joseph(约瑟夫)问题

时间限制: 2000ms; 空间限制: 65 536KB。

问题描述: Joseph 问题是众所周知的。有 n 个人, 编号为 $1, 2, \dots, n$, 站在一个圆圈中,

每隔 m 个人就杀一个人,最后仅剩一个人。Joseph 很聪明,可以选择最后一个人的位置,从而挽救他的生命。例如,当 $n=6$ 且 $m=5$ 时,按顺序出列的人员是 5,4,6,2,3,1,那么 1 会活下来。

假设在圈子里前面恰好有 k 个好人,后面恰好有 k 个坏人,则必须确定所有坏人都在第一个好人前面被杀的最小 m 。

输入格式:输入文件中包含若干行,每行一个 k ,最后一行为 0,可以假设 $0 < k \leq 14$ 。

输出格式:输出文件中每行给出输入文件中的 k 对应的最小 m 。

输入样例:

```
3
4
0
```

输出样例:

```
5
30
```

解: 题目中有多个测试实例,每个测试实例的结果仅仅与 k 有关系,设计避免重复计算的数组 a , $a[k]$ 记录 k 问题的 m 值, a 数组的所有元素初始为 0,当 $a[k]$ 不为 0 时说明该 k 问题已经求出,直接返回 $a[k]$ 即可。

对于每个 k 问题,假设 $2k$ 个人的序号是 $0 \sim 2k-1$,显然 m 从 $k+1$ 开始枚举(因为每次从头开始,若 $m < k+1$,则第一个被杀的一定是好人)。对于每个 m :

(1) cnt 表示剩余人数,所以 cnt 从 $2k$ 开始到 $k+1$ 循环, p 从 0 开始找被杀的人员的序号,每杀一个人, cnt 减少 1。

(2) 若被杀人员的序号 p 满足 $p < k$ (前 k 个好人的序号为 $0 \sim k-1$),说明 p 对应的是好人,置 $\text{cnt}=0$ 结束该 m 值的枚举。

(3) 若 $\text{cnt}=k$,则说明剩下 k 个人,并且被杀的 k 个人都是坏人,则找到这样的 m ,置 $a[k]=m$,返回 m 。

对应的 AC 程序如下:

```
import java.util.*;
public class Main
{
    static int[] ans=new int[15];           //记录 k 问题的 m 值,初始时均为 0
    public static int Joseph(int k)         //求解 Joseph 问题
    {
        int cnt,p;
        if(ans[k]!=0) return ans[k];        //若已经求出,直接返回
        for(int m=k+1;;m++)                //m 从 k+1 开始试探
        {
            for(cnt=2*k,p=0;cnt>k;cnt--)    //检查是否满足要求
            {
                p=(p+m-1)%cnt;             //被杀掉的人的位置
                if(p<k) cnt=0;               //被杀掉的人是前 k 个(好人)
            }
            if(cnt==k)                       //所有坏人都被杀了,满足要求
            {
                ans[k]=m;
                return m;
            }
        }
    }
}
```

```

    }
}

public static void main(String args[])
{
    Scanner cin=new Scanner(System.in);
    while(cin.hasNext())
    {
        int k=cin.nextInt();
        if(k==0) break;
        System.out.println(Joseph(k));
    }
}
}

```

上述程序的执行时间为 577ms,空间是 10 388KB。

3. POJ2389——公牛数学问题

时间限制: 1000ms; 空间限制: 65 536KB。

问题描述: 公牛在数学上比奶牛好多了,它们可以将巨大的整数相乘,得到完全精确的答案。农夫约翰想知道它们的答案是否正确,请帮助他检查公牛队的答案。读入两个正整数(每个不超过 40 位)并计算其结果,以常规方式输出(没有额外的前导零)。约翰要求不使用特殊的库函数进行乘法。

输入格式: 输入两行,每行包含一个十进制数。

输出格式: 输出一行表示乘积。

输入样例:

```

11111111111111
1111111111

```

输出样例:

```

12345679011110987654321

```

解: 两个长度最多为 40 位的正整数采用 String 对象 s1、s2 表示。设置 a、b 两个整数数组(初始时所有元素为 0),将 s1 的各位存放在 a 中(s1 的最后位存放在 a[0]中),将 s2 的各位存放在 b 中(s2 的最后位存放在 b[0]中)。

实现 a 与 b 相乘的过程是用 c 数组的 c[i+j]元素存放 a[i]和 b[j]相乘的累加值,然后调整有进位的元素,反向输出得到最后结果。

对应的 AC 程序如下:

```

import java.lang.*;
import java.util.*;

public class Main
{
    public static String mult(String s1,String s2)           //两个数字字符串相乘
    {
        int[] a=new int[42];
        int[] b=new int[42];
        int[] c=new int[90];
        int i,j;
        int m=s1.length();
        int n=s2.length();
    }
}

```

```

for(i=0;i<m;i++) //a[0]存放 s1 的最低位
    a[i]=s1.charAt(m-1-i)-'0';
for(i=0;i<n;i++) //b[0]存放 s2 的最低位
    b[i]=s2.charAt(n-1-i)-'0';
for(i=0;i<m;i++)
    for(j=0;j<n;j++)
        c[i+j] += a[i] * b[j];
for(i=0;i<80;i++)
    if(c[i] >= 10) //有进位
    {
        c[i+1] += c[i]/10;
        c[i] %= 10;
    }
String ans="";
for(i=c.length-1;i>=0;i--) //删除前导零
    if(c[i]!=0) break;
for(;i>=0;i--)
    ans += c[i];
return ans;
}

public static void main(String[] args)
{
    Scanner cin = new Scanner(System.in);
    String s1,s2;
    s1=cin.nextLine();
    s2=cin.nextLine();
    String s3=mult(s1,s2);
    System.out.println(s3);
}
}

```

上述程序的执行时间为 1266ms,空间是 2964KB。实际上,本题目可以直接使用 Java 的长整数类型来实现,对应的 AC 程序如下:

```

import java.math.BigInteger;
import java.util.Scanner;
public class Main
{
    public static void main(String[] args)
    {
        Scanner cin = new Scanner(System.in);
        BigInteger a = cin.nextBigInteger();
        BigInteger b = cin.nextBigInteger();
        a = a.multiply(b);
        System.out.println(a);
    }
}

```

上述程序的执行时间为 1375ms,空间是 3036KB。

说明: 尽管在线性表部分中讨论了多种存储结构,但在实际在线编程中,为了简单和提高效率往往直接采用数组代替顺序表,相反链表较少使用,其原因之一是在这类题目中都确定了数据量的大小,适合采用顺序结构,另外链表由于结点地址不连续,内存寻址空间较大,导致存取效率较低。

3.3 第3章 栈和队列

1. HDU1237——简单计算器

时间限制：2000ms；空间限制：65 536KB。

问题描述：读入一个只包含+、-、*、/的非负整数计算的表达式，计算该表达式的值。

输入格式：测试输入包含若干测试用例，每个测试用例占一行，每行不超过 200 个字符，整数和运算符之间用一个空格分隔；没有非法表达式；当一行中只有 0 时输入结束，相应的结果不要输出。

输出格式：对每个测试用例输出一行，即该表达式的值，精确到小数点后两位。

输入样例：

```
1 + 2
4 + 2 * 5 - 7 / 11
0
```

输出样例：

```
3.00
13.36
```

解：与《教程》中 3.1.7 节“用栈求解简单表达式求值问题”的原理相同，由于这里不包括括号，所以更简单一些，将求表达式值的两个步骤合并起来。

设计运算数栈为 opand，运算符栈为 oper。先将输入行 str 拆分为 strs 字符串数组，用 i 遍历 strs：

(1) 若 $\text{strs}[i]$ 为“*”或者“/”，为最高优先级，将 oper 栈顶为“*”或者“/”的运算符先计算，再将 $\text{strs}[i]$ 进 oper 栈。

(2) 若 $\text{strs}[i]$ 为“+”或者“-”，为最低优先级，将 oper 栈中所有的运算符先计算，再将 $\text{strs}[i]$ 进 oper 栈。

(3) 其他为运算数，将其进 opand 栈。

最后输出 opand 的栈顶数值。

对应的 AC 程序如下：

```
import java.util.*;
import java.util.Scanner;
public class Main
{
    static Stack<Double> opand=new Stack<Double>(); //运算数栈
    static Stack<String> oper=new Stack<String>(); //运算符栈
    public static boolean Same(String s1,String s2) //比较两个字符串值是否相同
    {
        return s1.compareTo(s2)==0;
    }
    public static void comp(String op) //用于一个运算符的计算
    {
        double a=opand.pop();
        double b=opand.pop();
        double c=0.0;
```

```

        if(Same(op, "+"))                //op 为“+”
            c=b+a;
        else if(Same(op, "-"))           //op 为“-”
            c=b-a;
        else if(Same(op, "*"))           //op 为“*”
            c=b*a;
        else if(Same(op, "/"))           //op 为“/”
            c=b/a;
        opand.push(c);                   //结果进 opand 栈
    }

    public static void main(String[] args)
    {
        Scanner fin = new Scanner(System.in);
        while(fin.hasNext())
        {
            String str=fin.nextLine();    //接收输入的字符串
            if(str.compareTo("0")==0)     //若是只输入 0,则结束
                break;
            opand.clear();
            oper.clear();
            String[] strs=str.split(" "); //根据题目要求分割字符串
            String op;
            double a,b,c;
            int i=0;
            while(i<strs.length)          //遍历 strs
            {
                if(Same(strs[i], "*") || Same(strs[i], "/")) //遇到 * 或者 /
                {
                    while(!oper.empty())
                    {
                        op=oper.peek();
                        if(Same(op, "*") || Same(op, "/")) //栈顶为 * 或者/,先计算
                        {
                            comp(op);
                            oper.pop();
                        }
                    }
                    else break;            //其他+或者-
                }
                oper.push(strs[i]);        //将 strs[i]进 oper 栈
            }
            else if(Same(strs[i], "+") || Same(strs[i], "-")) //遇到+或者-
            {
                while(!oper.empty())     //计算 oper 栈中所有的运算符
                {
                    op=oper.pop();
                    comp(op);
                }
                oper.push(strs[i]);       //将 strs[i]进 oper 栈
            }
            else                          //其他运算数进 opand 栈
                opand.push(Double.parseDouble(strs[i]));
            i++;
        }
        while(!oper.empty())             //计算 oper 栈中所有的运算符
        {
            op=oper.pop();
            comp(op);
        }
        System.out.printf("%.2f", opand.peek());
        System.out.println();
    }

```

```

    }
}
}

```

上述程序的执行时间为 452ms, 空间为 13 620KB。

说明: 类似的题目有 POJ3295。

2. POJ1363——铁轨问题

时间限制: 1000ms; 空间限制: 65 536KB。

问题描述: A 市有一个著名的火车站, 那里的山非常多。该站建于 20 世纪, 不幸的是该火车站是一个死胡同, 并且只有一条铁轨, 如图 3.1 所示。

当地的传统是从 A 方向到达的每列火车都继续沿 B 方向行驶, 需要以某种方式进行车厢重组。假设从 A 方向到达的火车有 n ($n \leq 1000$) 个车厢, 按照递增的顺序 1、2、 $\dots$ 、 n 编号。火车站负责人必须知道是否可以通过重组得到 B 方向的车厢序列。

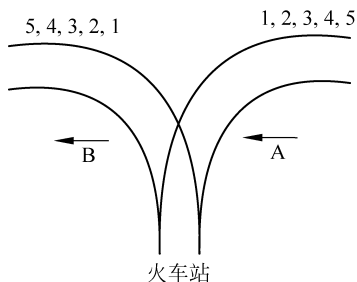


图 3.1 铁轨问题示意图

输入格式: 输入由若干行块组成; 除了最后一个之外, 每个块描述了一个列车以及可能更多的车厢重组要求; 在每个块的第一行中为上述的整数 n , 下一行是 1、2、 $\dots$ 、 n 的车厢重组序列; 每个块的最后一行仅包含 0; 最后一个块只包含一行 0。

输出格式: 输出包含与输入中具有车厢重组序列对应的行。如果可以得到车厢对应的重组序列, 输出一行“Yes”, 否则输出一行“No”。此外, 在输入的几个块之后有一个空行。

输入样例:

```

5
1 2 3 4 5
5 4 1 2 3
0
6
6 5 4 3 2 1
0
0

```

输出样例:

```

Yes
No

Yes

```