**常用算法题目**

1.算法是这样的，如果给定N个不同字符，将这N个字符全排列，最终的结果将会是N!种。如：给定 A、B、C三个不同的字符，则结果为：ABC、ACB、BAC、BCA、CAB、CBA一共3!=3\*2=6种情况。

**public** **class** AllPermutation

{

**public** **static** **void** main(String[] args)

{

//使用递归完成全排列

**char**[] source=**new** **char**[]{'A','B','C'};

**char**[] result=**new** **char**[source.length];

*allPermutation*(0,source,result);

}

/\*\*

\*

\* **@param** index当前考虑的数的下标(从0开始)

\* **@param** source

\* **@param** result

\*/

**public** **static** **void** allPermutation(**int** index,

**char**[] source,**char**[] result){

//当源数据中只有一个字符时，将该字符加入结果数组，并输出

**if**(source.length==1){

result[index]=source[0];

*show*(result);

**return** ;

}

**for**(**int** i=0;i<result.length-index;i++){

result[index]=source[i];

**char**[] newSource=*getNewSource*(source,source[i]);

*allPermutation*(index+1, newSource,result);

}

}

**public** **static** **void** show(**char**[] result){

System.*out*.println(result);

}

/\*\*

\* 生成去掉指定字符的新源数据数组

\* **@param** source 原来的源数据数组

\* **@param** c 指定去掉的字符

\* **@return**

\*/

**public** **static** **char**[] getNewSource(**char**[] source,**char** c){

**char**[] newSource=**new** **char**[source.length-1];

**for**(**int** i=0,j=0;i<source.length;i++){

**if**(source[i]!=c){

newSource[j]=source[i];

j++;

}

}

**return** newSource;

}

}

2. 串的简单处理

串的处理

在实际的开发工作中，对字符串的处理是最常见的编程任务。

本题目即是要求程序对用户输入的串进行处理。具体规则如下：

1. 把每个单词的首字母变为大写。

2. 把数字与字母之间用下划线字符（\_）分开，使得更清晰

3. 把单词中间有多个空格的调整为1个空格。

例如：

用户输入：

you and me what cpp2005program

则程序输出：

You And Me What Cpp\_2005\_program

用户输入：

**this** is a 99cat

则程序输出：

This Is A 99\_cat

我们假设：用户输入的串中只有小写字母，空格和数字，不含其它的字母或符号。

每个单词间由1个或多个空格分隔。

假设用户输入的串长度不超过200个字符。

**import** java.util.Scanner;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** Demo02 {

**public** **static** **void** print(String[] s){

**for**(**int** i=0;i<s.length-1;i++){

System.*out*.print(s[i]+" ");

}

System.*out*.println(s[s.length-1]);

}

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

Scanner scan = **new** Scanner(System.*in*);

String s = scan.nextLine();

String[] ss = s.split("[\\s]+");

**for**(**int** i=0;i<ss.length;i++){

// 大写

String up = (""+ss[i].charAt(0)).toUpperCase(); StringBuffer sb = **new** StringBuffer(ss[i]);

ss[i] = sb.replace(0, 1, up).toString();

Matcher m = Pattern.*compile*("\\d+").matcher(ss[i]);

**while**(m.find()){

String num = **new** String(m.group());

String num2 = num;

num2 = "\_"+num+"\_"; // 数字前添加"\_"

ss[i] = ss[i].replace(num, num2);

**if**(ss[i].startsWith("\_")){ // 去头"\_"

ss[i] = ss[i].substring(1);

}

**if**(ss[i].endsWith("\_")){ // 去尾"\_"

ss[i] = ss[i].substring(0,ss[i].length()-1);

}

}

}

*print*(ss);

}

}

运行结果:

you and me what cpp2005program

You And Me What Cpp\_2005\_program

3. 猜算式

看下面的算式：

□□ x □□ = □□ x □□□

它表示：两个两位数相乘等于一个两位数乘以一个三位数。

如果没有限定条件，这样的例子很多。

但目前的限定是：这9个方块，表示1~9的9个数字，不包含0。

该算式中1至9的每个数字出现且只出现一次！

比如：

46 x 79 = 23 x 158

54 x 69 = 27 x 138

54 x 93 = 27 x 186

.....

请编程，输出所有可能的情况！

注意：

左边的两个乘数交换算同一方案，不要重复输出！

不同方案的输出顺序不重要

**import** java.util.List;

**import** java.util.ArrayList;

**public** **class** Demo01 {

**static** List<String> *lis* = **new** ArrayList<String>(); // 结果1(有重复的结果)

**static** List<String> *lis2* = **new** ArrayList<String>(); // 结果2(去重复后的结果)

// 初始化数组为1~9

**public** **static** **void** init(**int**[] n) {

**for**(**int** i=0;i<9;i++){

n[i] = i+1; // 初始化数组为1~9

}

}

// 是否重复

**public** **static** **boolean** isDup(String s1,String s2){

String a1 = s1.substring(0,2);

String a2 = s1.substring(2,4);

String b1 = s2.substring(0,2);

String b2 = s2.substring(2,4);

**if**(a1.equals(b2)&&a2.equals(b1)){

**return** **true**;

}**else**{

**return** **false**;

}

}

// 去除lis重复元素

**public** **static** **void** removeDuplicate(){

*lis2*.add(*lis*.get(0));

**for**(**int** i=1;i<*lis*.size();i++){

**boolean** flag = **true**; // 标记是否重复

**for**(**int** j=0;j<*lis2*.size();j++){

flag = *isDup*(*lis*.get(i),*lis2*.get(j)); // 判断是否重复

// 如果元素重复,直接跳出这层循环,测试下个数据

**if**(flag) **break**;

}

**if**(!flag){

// 不重复,则添加

*lis2*.add(*lis*.get(i));

}

}

}

// 输出

**public** **static** **void** print(){

**for**(String s:*lis2*){

String a = s.substring(0,2);

String b = s.substring(2,4);

String c = s.substring(4,6);

String d = s.substring(6);

System.*out*.println(a+" x "+b+" = "+c+" x "+d);

}

}

// 检测结果,符合条件的输出

**public** **static** **void** check(**int**[] n){

StringBuffer sb = **new** StringBuffer();

**for**(**int** x:n){ // 数字数组转为字符串

sb.append(x);

}

**int** a = Integer.*parseInt*(sb.substring(0,2).toString());

**int** b = Integer.*parseInt*(sb.substring(2,4).toString());

**int** c = Integer.*parseInt*(sb.substring(4,6).toString());

**int** d = Integer.*parseInt*(sb.substring(6).toString());

**if**(a\*b==c\*d){

*lis*.add(sb.toString()); // 添加结果

}

}

// 全排列进行测试

**public** **static** **void** allSort(**int**[] n,**int** start,**int** end){

**if**(start>=end){

*check*(n); // 检测结果,符合条件的输出

**return** ;

}**else**{

**for**(**int** i=start;i<=end;i++){

**int** t = n[start]; // 交换元素

n[start] = n[i];

n[i] = t;

// 递归全排列

*allSort*(n,start+1,end);

t = n[start]; // 还原元素

n[start] = n[i];

n[i] = t;

}

}

}

**public** **static** **void** fun(){

**int**[] n = **new** **int**[9];

*init*(n); // 初始化数组为1~9

*allSort*(n,0,n.length-1); // 全排列测试

*removeDuplicate*(); // 去除重复元素

}

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

*fun*(); //

*print*(); // 输出结果

}

}

运行结果：

46 x 79 = 23 x 158

54 x 69 = 27 x 138

54 x 93 = 27 x 186

58 x 67 = 29 x 134

58 x 69 = 23 x 174

58 x 73 = 29 x 146

58 x 96 = 32 x 174

63 x 74 = 18 x 259

64 x 79 = 32 x 158

73 x 96 = 12 x 584

76 x 98 = 14 x 532

3.  Excel地址转换

Excel是最常用的办公软件。每个单元格都有唯一的地址表示。 比如：第12行第4列表示为：“D12”，第5行第255列表示为“IU5”。

事实上，Excel提供了两种地址表示方法，还有一种表示法叫做RC格式地址。

第12行第4列表示为：“R12C4”，第5行第255列表示为“R5C255”。

你的任务是：编写程序，实现从RC地址格式到常规地址格式的转换。

【输入、输出格式要求】

用户先输入一个整数n（n<100），表示接下来有n行输入数据。

接着输入的n行数据是RC格式的Excel单元格地址表示法。

程序则输出n行数据，每行是转换后的常规地址表示法。

例如：用户输入：

2

R12C4

R5C255

则程序应该输出：

D12

IU5

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Scanner;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** Demo03 {

// 计算出字母的个数

**public** **static** **int** checkCount(**int** n){

**int** count = 1;

**while**(**true**){

**int** t = (**int**)Math.*pow*(26, count);

**if**(n > t){

count++;

n -= t;

}**else**{

**return** count;

}

}

}

// 添加余数对应的字母

**public** **static** **char** f(**int** n){

**if**(n==26){

**return** 'Z';

}**else**{

**return** (**char**)('A'-1+n%26);

}

}

// 计算结果

**public** **static** String fun(**int** Row,**int** Col){

StringBuffer sb = **new** StringBuffer();

**int** count = *checkCount*(Col); // 计算出字母的个数

**while**(count>0){

**if**(Col%26==0){ //如果能除尽

// 例(702):702/26时(2余0) 应该分配成(1,26)

// 个位 加26('Z')时 就等于 十位上的2 去掉(1个26),

//(一个(个位的26)对应一个(十位的1))

// 修改n的值 2-1; n就等于1; 这时就分配成了(1,26);

sb.insert(0,'Z'); // 添加'Z'

Col = Col/26 -1;

}**else**{

sb.insert(0,*f*(Col));// 添加余数r对应的字母

Col /= 26; // 修改 n 的值

}

count--;

}

sb.append(Row); // RC地址后添加(行号)

**return** sb.toString();

}

// 输入数据

**public** **static** **void** input(String[] s){

Scanner scan = **new** Scanner(System.*in*);

**int** i = 0;

**while**(i<s.length){ // 输入n个数据

s[i++] = scan.nextLine();

}

}

// 拆分并计算结果

**public** **static** **void** splitOper(String[] s){

Pattern p = Pattern.*compile*("[0-9]+"); // 以数字做分隔

**for**(**int** i=0;i<s.length;i++){

Matcher m = p.matcher(s[i]); // 得到所有数字

m.find(); // 得到第一个数字

**int** Row = Integer.*parseInt*(m.group()); // 取出第一个数字

m.find(); // 得到第二个数字

**int** Col = Integer.*parseInt*(m.group()); // 取出第二个数字

System.*out*.println(*fun*(Row,Col)); // 计算结果并输出

}

}

// 主函数

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

Scanner scan = **new** Scanner(System.*in*);

System.*out*.println("输入一个整数n（n<100）");

String[] s = **new** String[scan.nextInt()];

*input*(s); // 输入数据

*splitOper*(s); // 拆分并计算结果

}

}

运行结果：

输入一个整数n（n<100）

2

R12C4

R5C255

D12

IU5

4. 手机尾号

/\*

30年的改革开放，给中国带来了翻天覆地的变化。2011全年中国手机产量约为11.72亿部。手机已经成为百姓的基本日用品！ 给手机选个好听又好记的号码可能是许多人的心愿。

但号源有限，只能辅以有偿选号的方法了。

这个程序的目的就是：根据给定的手机尾号（4位），按照一定的规则来打分。其规则如下：

1. 如果出现连号，不管升序还是降序，都加5分。例如：5678,4321都满足加分标准。

2. 前三个数字相同，或后三个数字相同，都加3分。例如：4888,6665,7777都满足加分的标准。

注意：7777因为满足这条标准两次，所以这条规则给它加了6分。

3. 符合AABB或者ABAB模式的加1分。例如：2255,3939,7777都符合这个模式，所以都被加分。

注意：7777因为满足这条标准两次，所以这条标准给它加了2分。

4. 含有：6，8，9中任何一个数字，每出现一次加1分。例如4326,6875,9918都符合加分标准。其中，6875被加2分；9918被加3分。

尾号最终得分就是每条标准的加分总和！

要求程序从标准输入接收数据，在标准输出上输出结果。

输入格式为：第一行是一个整数n（<100），表示下边有多少输入行，接下来是n行4位一组的数据，就是等待计算加分的手机尾号。

例如，输入：

14

3045

….

…..

6789

8866

则输出：

0

0

….

…

8

5

\*/

**import** java.util.Scanner;

**public** **class** Demo07 {

// 输入数据

**public** **static** **void** input(String[] n){

Scanner scan = **new** Scanner(System.*in*);

**int** i = 0;

**while**(i<n.length){

n[i++] = scan.nextLine();

}

}

// 得到分数

**public** **static** **int** getGrade(String n){

**int** s = 0; // 总分

**char**[] x = n.toCharArray();

**int** a = x[0]-'0';

**int** b = x[1]-'0';

**int** c = x[2]-'0';

**int** d = x[3]-'0';

**if**(a+1==b&&b+1==c&&c+1==d||a-1==b&&b-1==c&&c-1==d){

s += 5; // 1.出现连号 +5分

}

**if**(a==b&&b==c){

s += 3; // 2.三个数字相同 (前三)+3分

}

**if**(b==c&&c==d){

s += 3; // 2.三个数字相同 (后三)+3分

}

**if**(a==b&&c==d){

s += 1; // 3.AABB模式 +1分

}

**if**(a==c&&b==d){

s += 1; // 3.ABAB模式 +1分

}

**if**(a==6||a==8||a==9){

s += 1; // 4.a含有：6 或 8 或 9 +1分

}

**if**(b==6||b==8||b==9){

s += 1; // 4.b含有：6 或 8 或 9 +1分

}

**if**(c==6||c==8||c==9){

s += 1; // 4.c含有：6 或 8 或 9 +1分

}

**if**(d==6||d==8||d==9){

s += 1; // 4.d含有：6 或 8 或 9 +1分

}

**return** s;

}

// 取得每一行分数并输出

**public** **static** **void** fun(String[] n){

**for**(String x:n){

System.*out*.println(*getGrade*(x));

}

}

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

Scanner scan = **new** Scanner(System.*in*);

String[] n = **new** String[scan.nextInt()];

*input*(n); // 输入数据

*fun*(n);

}

}

运行结果：

14

3045

0211

…..

…..

……

8

5

5. 括号问题

下面的代码用于判断一个串中的括号是否匹配 所谓匹配是指不同类型的括号必须左右呼应，可以相互包含，但不能交叉

例如：

..(..[..]..).. 是允许的

..(...[...)....].... 是禁止的

对于 main 方法中的测试用例，应该输出：

false

true

false

false

请分析代码逻辑，并推测划线处的代码。

答案写在 “解答.txt” 文件中

注意：只写划线处应该填的内容，划线前后的内容不要抄写。

\*/

**import** java.util.\*;

**public** **class** Demo06 {

**public** **static** **boolean** isGoodBracket(String s) {

Stack<Character> a = **new** Stack<Character>();

**for** (**int** i = 0; i < s.length(); i++) {

**char** c = s.charAt(i);

**if** (c == '(')

a.push(')');

**if** (c == '[')

a.push(']');

**if** (c == '{')

a.push('}');

**if** (c == ')' || c == ']' || c == '}') {

**if** (a.size()==0)

**return** **false**; // 填空

**if** (a.pop() != c)

**return** **false**;

}

}

**if** (a.size()!=0)

**return** **false**; // 填空

**return** **true**;

}

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

System.*out*.println(*isGoodBracket*("...(..[.)..].{.(..).}..."));

System.*out*.println(*isGoodBracket*("...(..[...].(.).).{.(..).}..."));

System.*out*.println(*isGoodBracket*(".....[...].(.).){.(..).}..."));

System.*out*.println(*isGoodBracket*("...(..[...].(.).){.(..)...."));

}

}

运行结果：

false

true

false

false

6. 扑克牌移动

/\*

下面代码模拟了一套扑克牌（初始排序A~K，共13张）的操作过程。

操作过程是：

手里拿着这套扑克牌，从前面拿一张放在后面，再从前面拿一张放桌子上，再从前面拿一张放在后面，....

如此循环操作，直到剩下最后一张牌也放在桌子上。

下面代码的目的就是为了求出最后桌上的牌的顺序。

初始的排列如果是A,2,3...K，则最后桌上的顺序为：

[2, 4, 6, 8, 10, Q, A, 5, 9, K, 7, 3, J]

请分析代码逻辑，并推测划线处的代码。

答案写在 “解答.txt” 文件中

注意：只写划线处应该填的内容，划线前后的内容不要抄写。

\*/

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Vector;

**public** **class** Demo05 {

**public** **static** List moveCard(List src) {

**if** (src == **null**)

**return** **null**;

List dst = **new** Vector();

**for** (;;) {

**if** (src.size()==0)

**break**; // 填空

src.add(src.remove(0));

dst.add(src.remove(0)); // 填空

}

**return** dst;

}

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

List a = **new** Vector();

a.addAll(Arrays.*asList*("A", "2", "3", "4", "5", "6", "7", "8", "9",

"10", "J", "Q", "K"));

System.*out*.println(*moveCard*(a));

}

}

运行结果：

[2, 4, 6, 8, 10, Q, A, 5, 9, K, 7, 3, J]

7. 第一个数字

/\*

以下的静态方法实现了：把串s中第一个出现的数字的值返回。

如果找不到数字，返回-1

例如：

s = "abc24us43" 则返回2

s = "82445adb5" 则返回8

s = "ab" 则返回-1

请分析代码逻辑，并推测划线处的代码。

答案写在 “解答.txt” 文件中

注意：只写划线处应该填的内容，划线前后的内容不要抄写。

\*/

**public** **class** Demo04 {

**public** **static** **int** getFirstNum(String s) {

**if** (s == **null** || s.length() == 0)

**return** -1;

**char** c = s.charAt(0);

**if** (c >= '0' && c <= '9')

**return** s.charAt(0)-'0'; // 填空

**return** *getFirstNum*(s.substring(1)); // 填空

}

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

String s1 = "abc24us43"; //则返回2

String s2 = "82445adb5"; //则返回8

String s3 = "ab"; //则返回-1

System.*out*.println(*getFirstNum*(s1));

System.*out*.println(*getFirstNum*(s2));

System.*out*.println(*getFirstNum*(s3));

}

}

运行结果：

2

8

-1

8. 放麦子

/\*

你一定听说过这个故事。国王对发明国际象棋的大臣很佩服，

问他要什么报酬，大臣说：请在第1个棋盘格放1粒麦子，

在第2个棋盘格放2粒麦子，在第3个棋盘格放4粒麦子，

在第4个棋盘格放8粒麦子，......后一格的数字是前一格的两倍，

直到放完所有棋盘格（国际象棋共有64格）。

国王以为他只是想要一袋麦子而已，哈哈大笑。

当时的条件下无法准确计算，但估算结果令人吃惊：即使全世界都铺满麦子也不够用！

请你借助计算机准确地计算，到底需要多少粒麦子。

答案写在“解答.txt”中，不要写在这里！

\*/

**import** java.math.BigInteger;

**public** **class** Demo03 {

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

BigInteger bi = **new** BigInteger("0");

**for**(**int** i=0;i<64;i++){

bi = bi.add(BigInteger.*valueOf*((**long**)(Math.*pow*(2, i))));

}

System.*out*.println(bi);

}

}

运行结果：

18446744073709551614

/\*

\* 求21位数的水仙花数

\*/

**import** java.math.BigInteger;

**public** **class** Demo01\_BigInteger {

// 求 每个 i 的 21 次方

**public** **static** BigInteger p(**int** i){

BigInteger base = BigInteger.*valueOf*(i);

**return** base.pow(21);

}

**public** **static** **void** ji\_suan(BigInteger[] pw,**int**[] nn){

BigInteger sum = BigInteger.*ZERO*;

**for**(**int** i=0;i<10;i++){

sum = sum.add(

pw[i].multiply(BigInteger.*valueOf*(nn[i])));

}

String s = "" + sum;

**if**(s.length()!=21) **return**;

// 确定各数字出现的多少次

**int**[] nn2 = **new** **int**[10];

**for**(**int** i=0;i<21;i++){

nn2[s.charAt(i)-'0']++;

}

**for**(**int** i=0;i<10;i++){

**if**(nn[i]!=nn2[i]) **return**;

}

System.*out*.println(s);

}

**public** **static** **void** f(BigInteger[] pw, **int**[] nn, **int** cur, **int** use){

**if**(cur==9){

nn[9] = 21 - use;

*ji\_suan*(pw,nn);

**return**;

}

// 对当前位置所有可能进行枚举

**for**(**int** i=0;i<21-use;i++){

nn[cur] = i;

*f*(pw,nn,cur+1,use+i);

}

}

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

**long** startTime = System.*currentTimeMillis*(); // 程序开始时间

BigInteger pw[] = **new** BigInteger[10];

**for**(**int** i=0;i<pw.length;i++){

pw[i] = *p*(i);

}

**int** nn[] = **new** **int**[10];

*f*(pw,nn,0,0);

System.*out*.println("OK");

**long** endTime = System.*currentTimeMillis*(); // 程序结束时间

System.*out*.println((endTime-startTime)/1000f+"秒"); // 运行总时间

}

}

运行结果：

128468643043731391252

449177399146038697307

OK

27.343秒

8.猜生日

/\*

今年的植树节（2012年3月12日），小明和他的叔叔还有小伙伴们一起去植树。

休息的时候，小明的同学问他叔叔多大年纪，他叔叔说：“我说个题目，看你们谁先猜出来！”

“把我出生的年月日连起来拼成一个8位数（月、日不足两位前补0）正好可以被今天的年、月、日整除！”

他想了想，又补充到：“再给个提示，我是6月出生的。”

根据这些信息，请你帮小明算一下，他叔叔的出生年月日。

答案写在“解答.txt”中，不要写在这里！

格式是年月日连成的8位数。

例如，如果是1948年6月12日，就写：19480612

\*/

**public** **class** Demo02 {

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

**int** n = 20120312;

String s = "";

**for**(**int** i=19000601;i<=20130312;i++){

s = ""+i;

**int** d = Integer.*parseInt*(s.substring(6)); // 得到天

**int** m = Integer.*parseInt*(s.substring(4,6)); // 得到月

**if**(d==0) **continue**;

**if**(d>=1&&d<31&&m==6&&i%2012==0&&i%3==0&&i%12==0){

System.*out*.println(i);

}

}

}

}

运行结果：

19550604

9.填算式

/\*

看这个算式：

☆☆☆ + ☆☆☆ = ☆☆☆

如果每个五角星代表 1 ~ 9 的不同的数字。

这个算式有多少种可能的正确填写方法？

173 + 286 = 459

295 + 173 = 468

173 + 295 = 468

183 + 492 = 675

以上都是正确的填写法！

注意：

111 + 222 = 333 是错误的填写法！

因为每个数字必须是不同的！

也就是说：1~9中的所有数字，每个必须出现且仅出现一次！

注意：

不包括数字“0”！

注意：

满足加法交换率的式子算两种不同的答案。

所以答案肯定是个偶数！

注意：

只要求计算不同的填法的数目

不要求列出所有填写法

更不要求填写源代码！

\*/

**public** **class** Demo01 {

**static** **int** *count*=0;

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

**for**(**int** i=1;i<=9;i++){ // 第一个数的百位

**for**(**int** j=0;j<=9;j++){ // 十位

**for**(**int** k=0;k<=9;k++){ // 个位

**for**(**int** l=1;l<=9;l++){ // 第二个数的百位

**for**(**int** m=0;m<=9;m++){ // 十位

**for**(**int** n=0;n<=9;n++){ // 个位

**if**(i!=j && i!=k && i!=l && i!=m && i!=n &&

j!=k && j!=l && j!=m && j!=n &&

k!=l && k!=m && k!=n &&

l!=m && l!=n &&

m!=n ){

**int** a = i\*100+j\*10+k;

**int** b = l\*100+m\*10+n;

**int** c = a + b;

**int** x,y,z;

**if**(c>99&&c<999){

x = c/100;

y = c%100/10;

z = c%10;

}**else**{

**break**;

}

**if**(x!=y&&x!=z&&y!=z &&

x!=i && x!=j && x!=k && x!=l && x!=m && x!=n &&

y!=i && y!=j && y!=k && y!=l && y!=m && y!=n &&

z!=i && z!=j && z!=k && z!=l && z!=m && z!=n ){

System.*out*.println(a+" + "+b+" = "+c);

*count*++;

}

}

}

}

}

}

}

}

System.*out*.println("总个数:"+*count*);

}

}

运行结果：

103 + 469 = 572

….

…..

…..

847 + 106 = 953

总个数:1088

10. 火柴游戏

/\*

【编程题】(满分34分)

这是一个纵横火柴棒游戏。如图[1.jpg]，在3x4的格子中，游戏的双方轮流放置火柴棒。其规则是：

1. 不能放置在已经放置火柴棒的地方（即只能在空格中放置）。

2. 火柴棒的方向只能是竖直或水平放置。

3. 火柴棒不能与其它格子中的火柴“连通”。所谓连通是指两根火柴棒可以连成一条直线，

且中间没有其它不同方向的火柴“阻拦”。

例如：图[1.jpg]所示的局面下，可以在C2位置竖直放置（为了方便描述格子位置，图中左、下都添加了标记），

但不能水平放置，因为会与A2连通。同样道理，B2，B3，D2此时两种方向都不可以放置。但如果C2竖直放置后，

D2就可以水平放置了，因为不再会与A2连通（受到了C2的阻挡）。

4. 游戏双方轮流放置火柴，不可以弃权，也不可以放多根。直到某一方无法继续放置，则该方为负（输的一方）。

游戏开始时可能已经放置了多根火柴。

你的任务是：编写程序，读入初始状态，计算出对自己最有利的放置方法并输出。

如图[1.jpg]的局面表示为：

00-1

-000

0100

即用“0”表示空闲位置，用“1”表示竖直放置，用“-”表示水平放置。

【输入、输出格式要求】

用户先输入整数 n(n<100), 表示接下来将输入 n 种初始局面，每种局面占3行(多个局面间没有空白行)。

程序则输出：每种初始局面情况下计算得出的最佳放置法（行号+列号+放置方式）。

例如：用户输入：

2

0111

-000

-000

1111

----

0010

则程序可以输出：

00-

211

不难猜出，输出结果的含义为：

对第一个局面，在第0行第0列水平放置

对第二个局面，在第2行第1列垂直放置

注意：

行号、列号都是从0开始计数的。

对每种局面可能有多个最佳放置方法（解不唯一），只输出一种即可。

例如，对第一个局面，001 也是正解；最第二个局面，201也是正解。

\*/

**import** java.util.Scanner;

**import** java.util.List;

**import** java.util.ArrayList;

**public** **class** Demo05 {

// 用来标记是否连通

**static** **boolean** *flag* = **false**;

// 用来标记是否没有结果, 如果没有结果输出"空";

**static** **boolean** *flag2* = **false**;

// 初始化数组

**public** **static** **void** init(List<**char**[][]> lis,String[] s,**int** n){

**for**(**int** i=0;i<n;i++){

**for**(**int** j=i\*3;j<i\*3+3;j++){

lis.get(i)[j%3] = s[j].toCharArray();

}

}

}

// 创建n个数组 初始化,并存入lis

**public** **static** **void** input(List<**char**[][]> lis,**int** n){

Scanner scan = **new** Scanner(System.*in*);

**for**(**int** i=1;i<=n;i++){ // 创建数组

lis.add(**new** **char**[3][4]);

}

String[] s = **new** String[n\*3];

**for**(**int** i=0;i<n\*3;i++){ // 行到输入的数据

s[i] = scan.nextLine();

}

*init*(lis,s,n); // 用输入的数据 初始化每个数组

}

// c='1' 检查列上侧是否连通

**public** **static** **boolean** colU(**char**[][] m,**int** i,**int** j,**char** c){

**if**(i<0){

*flag* = **true**; // 都不连通

**return** *flag*;

}

**if**(m[i][j]=='0'){

**return** *colU*(m,i-1,j,c);

}**else** **if**(m[i][j]=='1'){

*flag* = **false**; // 有一个 '1' 则连通

**return** *flag*;

}**else** **if**(m[i][j]=='-'){

*flag* = **true**; // 有一个 '-' 则不连通

**return** *flag*;

}

**return** *flag*;

}

// c='1' 检查列下侧是否连通

**public** **static** **boolean** colD(**char**[][] m,**int** i,**int** j,**char** c){

**if**(i>=m.length){

*flag* = **true**; // 都不连通

**return** *flag*;

}

**if**(m[i][j]=='0'){

**return** *colD*(m,i+1,j,c);

}**else** **if**(m[i][j]=='1'){

*flag* = **false**; // 有一个 '1' 则连通

**return** *flag*;

}**else** **if**(m[i][j]=='-'){

*flag* = **true**; // 有一个 '-' 则不连通

**return** *flag*;

}

**return** *flag*;

}

// c='-' 检查行左侧是否连通

**public** **static** **boolean** rowL(**char**[][] m,**int** i,**int** j,**char** c){

**if**(j<0){

*flag* = **true**; // 都不连通

**return** *flag*;

}

**if**(m[i][j]=='0'){

**return** *rowL*(m,i,j-1,c);

}**else** **if**(m[i][j]=='1'){

*flag* = **true**; // 有一个 '1' 则不连通

**return** *flag*;

}**else** **if**(m[i][j]=='-'){

*flag* = **false**; // 有一个 '-' 则连通

**return** *flag*;

}

**return** *flag*;

}

// c='-' 检查行右侧是否连通

**public** **static** **boolean** rowR(**char**[][] m,**int** i,**int** j,**char** c){

**if**(j>=m[i].length){

*flag* = **true**; // 都不连通

**return** *flag*;

}

**if**(m[i][j]=='0'){

**return** *rowR*(m,i,j+1,c);

}**else** **if**(m[i][j]=='1'){

*flag* = **true**; // 有一个 '1' 则不连通

**return** *flag*;

}**else** **if**(m[i][j]=='-'){

*flag* = **false**; // 有一个 '-' 则连通

**return** *flag*;

}

**return** *flag*;

}

// 当c='1'时 检查是否连通1111111111111111111

**public** **static** **boolean** check1(**char**[][] m, **int** i, **int** j, **char** c) {

// 是 '1' 时 检查(上下)是否连通

**if**(*colU*(m,i,j,c)&&*colD*(m,i,j,c)){

*flag* = **true**;

}**else**{

**return** **false**;

}

**return** *flag*;

}

// 当c='-'时 检查是否连通-------------------

**public** **static** **boolean** check2(**char**[][] m, **int** i, **int** j, **char** c) {

// 是 '-' 时 检查(左右)是否连通

**if**(*rowL*(m,i,j,c)&&*rowR*(m,i,j,c)){

*flag* = **true**;

}**else**{

**return** **false**;

}

**return** *flag*;

}

// 检测并添加元素

**public** **static** **void** calc2(**char**[][] m,**int** i,**int** j){

**if**(*check1*(m, i, j, '1')){ // 等于'1'时,行列都不连通

m[i][j] = '1'; // 则添加元素

}**else** **if**(*check2*(m, i, j, '-')){// 等于'-'时,行列都不连通

m[i][j] = '-';// 则添加元素

}**else**{

*flag2* = **true**; // 表示无结果

}

}

// 计算函数的入口

**public** **static** **void** calc(**char**[][] m){

**for**(**int** i=0;i<m.length;i++){

**for**(**int** j=0;j<m[i].length;j++){

**if**(m[i][j]=='0'){

*calc2*(m,i,j); // 进入检测函数

}

**if**(*flag*){

String temp = i+""+j+""+m[i][j];

System.*out*.println(temp);

// 如果有元素添加,那么就不为空!下边的就不必输出"空"值

*flag2* = **false**;

**break**; // 如果添加过元素,则退出循环

}

}

**if**(*flag*){ // 如果添加过元素,则退出循环

*flag* = **false**;

**break**;

}

}

**if**(*flag2*){ // 如果无结果,则添加空

System.*out*.println("空");

*flag2* = **false**; // 修改标记位

}

}

// 主函数

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

Scanner scan = **new** Scanner(System.*in*);

List<**char**[][]> lis = **new** ArrayList<**char**[][]>(); // 存放每个数组

System.*out*.print("输入整数 n(n<100):");

// 创建n个数组 初始化,并存入lis

*input*(lis,scan.nextInt());

// 计算函数的入口, 单独去每个数组去测试

**for**(**int** i=0;i<lis.size();i++){

*calc*(lis.get(i));

}

}

}

11. 古代赌局

/\*

\* 【编程题】(满分23分)

俗话说：十赌九输。因为大多数赌局的背后都藏有阴谋。不过也不尽然，有些赌局背后藏有的是：“阳谋”。

有一种赌局是这样的：桌子上放六个匣子，编号是1至6。多位参与者（以下称玩家）可以把任意数量的钱押在某个编号的匣子上。

所有玩家都下注后，庄家同时掷出3个骰子（骰子上的数字都是1至6）。输赢规则如下：

1. 若某一个骰子上的数字与玩家所押注的匣子号相同，则玩家拿回自己的押注，庄家按他押注的数目赔付（即1比1的赔率）。

2. 若有两个骰子上的数字与玩家所押注的匣子号相同，则玩家拿回自己的押注，庄家按他押注的数目的2倍赔付（即1比2的赔率）。

3. 若三个骰子上的数字都与玩家所押注的匣子号相同，则玩家拿回自己的押注，庄家按他押注的数目的6倍赔付（即1比6的赔率）。

4. 若玩家所押注匣子号与某个骰子示数乘积等于另外两个骰子示数的乘积，则玩家拿回自己的押注，庄家也不赔付（流局）。

5. 若以上规则有同时满足者，玩家可以选择对自己最有利的规则。规则执行后，则庄家收获所有匣子上剩余的押注。

乍一看起来，好像规则对玩家有利，庄家吃亏。但经过大量实战，会发现局面很难说，于是怀疑是否庄家做了手脚，

庄家则十分爽快地说：可以由玩家提供骰子，甚至也可以由玩家来投掷骰子。

你的任务是：通过编程模拟该过程。模拟50万次，假定只有1个玩家，他每次的押注都是1元钱，其押注的匣子号是随机的。

再假定庄家有足够的资金用于赔付。最后计算出庄家的盈率（庄家盈利金额/押注总金额）。

【输入、输出格式要求】

程序无输入，程序运行输出庄家的盈率，四舍五入保留到小数后3位。

【注意】

请仔细调试！您的程序只有能运行出正确结果的时候才有机会得分！

请把所有类写在同一个文件中，调试好后，存入与【考生文件夹】下对应题号的“解答.txt”中即可。

相关的工程文件不要拷入。

请不要使用package语句。

源程序中只能出现JDK1.5中允许的语法或调用。不能使用1.6或更高版本。

\*/

**import** java.math.BigDecimal;

**public** **class** Demo04 {

**static** **int** *sum* = 0; // 庄家总钱数

**public** **static** **void** f(**int** m,**int** n,**int** a,**int** b,**int** c){

**if**(n==a&&n==b&&n==c){

*sum* -= 6\*m;

}**else** **if**(n==a&&n==b||n==a&&n==c||n==b&&n==c){

*sum* -= 2\*m;

}**else** **if**(n==a||n==b||n==c){

*sum* -= m;

}**else** **if**(!(n\*a==b\*c||n\*b==a\*c||n\*c==a\*b)){

*sum* += m;

}

}

**public** **static** **void** main(String[] args) **throws** Exception{

**int** m = 1; // 押注都是1元钱

**int** n; // 押注匣子号

**int** a; // 第一个骰子

**int** b; // 第二个骰子

**int** c; // 第三个骰子

**for**(**int** i=0;i<500000;i++){

n = (**int**)(Math.*random*()\*6+1);

a = (**int**)(Math.*random*()\*6+1);

b = (**int**)(Math.*random*()\*6+1);

c = (**int**)(Math.*random*()\*6+1);

*f*(m,n,a,b,c);

}

**double** d = *sum*/500000f;

System.*out*.printf("%.3f\n",d);

}

}

12. 源码变换

/\*

超文本标记语言（即HTML），是用于描述网页文档的一种标记语言。

HTML通过文本来描述文档显示出来应该具有的“样子”。它主要通过标签来定义对象的显示属性或行为。

如果把java的源文件直接拷贝到HTML文档中，用浏览器直接打开，会发现本来整齐有序的源文件变成了一团遭。

这是因为，文本中的许多回车和空格都被忽略了。而有些符号在html中有特殊的含义，引起了更复杂的局面。

为了源文件能正常显示，我们必须为文本加上适当的标签。对特殊的符号进行转义处理。

常用的有：

HTML 需要转义的实体：

& ---> &

空格 ---> &nbsp;

< ---> <

> ---> >

" ---> "

此外，根据源码的特点，可以把 TAB 转为4个空格来显示。

TAB ---> &nbsp;&nbsp;&nbsp;&nbsp;

为了显示为换行，需要在行尾加<br/>标签。

为了显示美观，对关键字加粗显示，即在关键字左右加<b>标签。比如：

<b>public</b>

对单行注释文本用绿色显示，可以使用<font>标签，形如：

<font color=green>//这是我的单行注释！</font>

注意：如果“//”出现在字符串中，则注意区分，不要错误地变为绿色。

不考虑多行注释的问题 /\* .... \*/ /\*或\*/ /\*\* .... \*/

/\*

你的任务是：编写程序，把给定的源文件转化为相应的html表达。

【输入、输出格式要求】

与你的程序同一目录下，存有源文件 a.txt，其中存有标准的java源文件。要求编写程序把它转化为b.html。

例如：目前的 a.txt 文件与 b.html 文件就是对应的。可以用记事本打开b.html查看转换后的内容。用浏览器打开b.html则可以看到显示的效果。

注意：实际评测的时候使用的a.txt与示例是不同的。

\*/

**import** java.io.BufferedReader;

**import** java.io.FileReader;

**import** java.io.PrintWriter;

**public** **class** Demo03 {

**public** **static** String process(String s){

StringBuffer sb = **new** StringBuffer();

**char**[] c = s.toCharArray();

**for**(**int** i=0;i<c.length;i++){

**switch**(c[i]){

**case** '&': sb.append("&"); **break**;

**case** ' ': sb.append("&nbsp;"); **break**;

**case** '<': sb.append("<"); **break**;

**case** '>': sb.append(">"); **break**;

**case** '"': sb.append("""); break;

**case** '\t': sb.append("&nbsp;&nbsp;&nbsp;&nbsp;"); **break**;

**default**: sb.append(c[i]);

}

}

String t = sb.toString();

t = t.replaceAll("public", "<b>public</b>");

t = t.replaceAll("class", "<b>class</b>");

t = t.replaceAll("static", "<b>static</b>");

t = t.replaceAll("void", "<b>void</b>");

**int** n = t.lastIndexOf("//"); // 找到最后边的 "//" 位置

**if**(n!=-1){

String temp = t.substring(n); //为注释 加绿色

t = t.replaceAll(temp, "<font color=green>"+temp+"</font>");

}

t = t+"<br/>\n";

**return** t;

}

**public** **static** **void** main(String[] args) **throws** Exception{

// 定义读取内容

BufferedReader br = **new** BufferedReader(**new** FileReader("a.txt"));

// 定义输出内容

PrintWriter pw = **new** PrintWriter("b.html");

String s;

pw.write("<html><body>\n");

**while**((s=br.readLine())!=**null**){

pw.write(*process*(s)); // 输出内容

}

pw.write("</body></html>");

br.close();

pw.close();

System.*out*.println("转换成功");

}

}

运行结果：

// 我的工具类

public class MyTool

{

public static void main(String[] args)

{

int a = 100;

int b = 20;

if(a>b && true)

System.out.println(a);

else

System.out.println("this! //aaa//kkk"); // 测试注释显示是否正确

}

}

13. 数量周期

/\*

复杂现象背后的推动力，可能是极其简单的原理。科学的目标之一就是发现纷

繁复杂的自然现象背后的简单法则。爱因斯坦的相对论是这方面的典范例证。

很早的时候，生物学家观察某区域某种昆虫的数量（称为虫口数）之逐年变化规律，

就十分迷惑：有的时候是逐渐增多达到一个平衡值。有的时候在两个数字间周期跳动。

有的时候则进入一片混乱，类似随机数字一样变化（称为混沌现象）。

慢慢地，人们从数学中更清晰地观察到了这一现象，并因此开创了：符号动力学、非线性动力学等研究领域。

一个著名的虫口数目简化模型如下：

x' = x \* (1 - x) \* r

这里，x x' r 都是浮点数。

其中，x 表示当年的虫口数，x' 表示下一年的虫口数。

它们的取值范围在 0 与 1 之间，实际上表示的是：虫口的总数占环境所能支持的最大数量的比率。

r 是常数（环境参数），r的取值范围在 [0,4]。

令人惊讶的是：这个简单的迭代公式有着不同寻常的神秘性质！

一般来说，多次迭代后，虫口数的稳定模式与x的初始值无关，而与 r 有关！

例如：无论x初始值是多少，当 r = 2.5 的时候，x 多次迭代后会趋向于 0.6。

而当 r = 3.2 的时候，x 的值会趋向于在 0.799 与 0.513 之间周期性摆动。

那么，r = 3.62 的时候，你观察到有什么周期现象发生吗？

不需要提交源代码，只要写出你的结论即可！

答案写在：“解答.txt”中，不要写在这里。

\*/

**public** **class** Demo01 {

**static** **int** *count* = 100; // 执行100次退出

**public** **static** **void** f(**double** x,**double** r){

**if**(*count*<=0) **return**;

x = x \* (1 - x) \* r;

System.*out*.println(x);

*count*--;

*f*(x,r);

}

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

**double** x = 0.2;

**double** r = 3.62;

*f*(x,r);

System.*out*.println("网络某某结论：虫口数目函数呈锯齿状变化，" +

"虫口数目不存在连续两年增加和连续两年减少的情况。");

}

}

运行结果：

0.5792000000000002

……

……

……

0.878401825611548

网络某某结论：虫口数目函数呈锯齿状变化，虫口数目不存在连续两年增加和连续两年减少的情况。

14. 提取子串

/\*

串“abcba”以字母“c”为中心左右对称；串“abba” 是另一种模式的左右对称。

这两种情况我们都称这个串是镜像串。特别地，只含有1个字母的串，可以看成是第一种模式的镜像串。

一个串可以含有许多镜像子串。我们的目标是求一个串的最大镜像子串（最长的镜像子串），

如果有多个最大镜像子串，对称中心靠左的优先选中。例如：“abcdeefghhgfeiieje444k444lmn”的最大镜像子串是：“efghhgfe”

下面的静态方法实现了该功能，请仔细阅读并分析代码，填写空白处的代码，使得程序的逻辑合理，结果正确。

方法一：

\*/

**public** **class** Demo02\_two {

**public** **static** String maxS(String s){

String maxS = "";

**char**[] c = s.toCharArray();

**for**(**int** i=0;i<c.length-1;i++){

**int** mark = 0; // 下标标记

String temp = ""; // 记录一个镜像串

**if**(c[i]==c[i+1]){

**for**(;;){ // abba模式

**if**((i-mark<0)||c[i-mark]!=c[i+mark+1]) **break**;

mark++;

}

temp = s.substring(i-mark+1,i+mark+1);

}**else** **if**((i+2)<c.length&&c[i]==c[i+2]){

**for**(;;){ // abcba模式

**if**((i-mark<0)||c[i-mark]!=c[i+mark+2]) **break**;

mark++;

}

temp = s.substring(i-mark+1,i+mark+2);

}

**if**(temp.length()>maxS.length()){

maxS = temp;

}

}

**return** maxS;

}

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

String s = "abcdeefghhgfeiieje444k444lmn";

System.*out*.println(*maxS*(s));

}

}

方法二：

**public** **class** Demo02 {

**public** **static** String getMaxMirrorString(String s) {

String max\_s = ""; // 所求的最大对称子串

**for** (**int** i = 0; i < s.length(); i++) {

// 第一种对称模式

**int** step = 1;

**try** {

**for** (;;) {

**if** (s.charAt(i - step) != s.charAt(i + step))

**break**;

step++;

}

} **catch** (Exception e) {

}

String s1 = s.substring(i - step + 1, i + step); // 填空1

// 第二种对称模式

step = 0;

**try** {

**for** (;;) {

**if** (s.charAt(i - step) != s.charAt(i + step + 1))

**break**; // 填空2

step++;

}

} **catch** (Exception e) {

}

String s2 = s.substring(i - step + 1, i + step + 1);

**if** (s1.length() > max\_s.length())

max\_s = s1;

**if** (s2.length() > max\_s.length())

max\_s = s2;

}

**return** max\_s;

}

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

String s = "abcdeefghhgfeiieje444k444lmn";

System.*out*.println(*getMaxMirrorString*(s));

}

}

运行结果：

efghhgfe

15. 取球游戏

/\*

今盒子里有n个小球，A、B两人轮流从盒中取球，每个人都可以看到另一个人取了多少个，

也可以看到盒中还剩下多少个，并且两人都很聪明，不会做出错误的判断。

我们约定：

每个人从盒子中取出的球的数目必须是：1，3，7或者8个。

轮到某一方取球时不能弃权！

A先取球，然后双方交替取球，直到取完。

被迫拿到最后一个球的一方为负方（输方）

请编程确定出在双方都不判断失误的情况下，对于特定的初始球数，A是否能赢？

程序运行时，从标准输入获得数据，其格式如下：

先是一个整数n(n<100)，表示接下来有n个整数。然后是n个整数，每个占一行（整数<10000），表示初始球数。

程序则输出n行，表示A的输赢情况（输为0，赢为1）。

例如，用户输入：

4

1

2

10

18

则程序应该输出：

0

1

1

0

\*/

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Scanner;

**public** **class** Demo06 {

**public** **static** **boolean** contest(**int** n){

**if**(n>=1){

**switch**(n){

**case** 1 : **return** **false**; // 剩1个球,则输

**case** 3 : **return** **false**; // 剩3个球,则输

**case** 7 : **return** **false**; // 剩7个球,则输

**case** 8 : **return** **true**; // 剩8个球,则赢

// 如果不是不是1，3，7，8则 选择权交给B，B任然调用该函数，不过返回值需要取反

// 而此时A可以选的只有 1 3 7 8 所以用num减去之 逐个测试即可

**default** : **return** (!*contest*(n-8)||!*contest*(n-7)||

!*contest*(n-3)||!*contest*(n-1));

}

}**else**{

**return** **false**;

}

}

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

Scanner scan = **new** Scanner(System.*in*);

**int** n = scan.nextInt();

List<Integer> lis = **new** ArrayList<Integer>();

**for**(**int** i=0;i<n;i++){ // 输入数据

lis.add(scan.nextInt());

}

System.*out*.println("输出结果:");

**for**(**int** i=0;i<n;i++){

**if**(*contest*(lis.get(i))){

System.*out*.println(1);

}**else**{

System.*out*.println(0);

}

}

}

}

运行结果：

4

1

2

10

18

输出结果:

0

1

1

0

16. 密码发生器

/\*

在对银行账户等重要权限设置密码的时候，我们常常遇到这样的烦恼：如果为了好记用生日吧，

容易被破解，不安全；如果设置不好记的密码，又担心自己也会忘记；如果写在纸上，担心纸张被别人发现或弄丢了...

这个程序的任务就是把一串拼音字母转换为6位数字（密码）。

我们可以使用任何好记的拼音串(比如名字，王喜明，就写：wangximing)作为输入，程序输出6位数字。

变换的过程如下：

第一步. 把字符串6个一组折叠起来，比如wangximing则变为：

wangxi

ming

第二步. 把所有垂直在同一个位置的字符的ascii码值相加，得出6个数字，如上面的例子，则得出：

228 202 220 206 120 105

第三步. 再把每个数字“缩位”处理：就是把每个位的数字相加，得出的数字如果不是一位数字，

就再缩位，直到变成一位数字为止。例如: 228 => 2+2+8=12 => 1+2=3

上面的数字缩位后变为：344836, 这就是程序最终的输出结果！

要求程序从标准输入接收数据，在标准输出上输出结果。

输入格式为：第一行是一个整数n（<100），表示下边有多少输入行，接下来是n行字符串，就是等待变换的字符串。

输出格式为：n行变换后的6位密码。

例如，输入：

5

zhangfeng

wangximing

jiujingfazi

woaibeijingtiananmen

haohaoxuexi

则输出：

772243

344836

297332

716652

875843

\*/

**import** java.util.Scanner;

**public** **class** Demo04 {

// 缩位

**public** **static** **char** con(**char**[] c){

**if**(c.length<=1){

**return** c[0];

}

**int** temp = 0;

**for**(**int** i=0;i<c.length;i++){

temp += c[i]-'0';

}

c = (""+temp).toCharArray();

**return** *con*(c);

}

// 缩位处理,并输出结果

**public** **static** **void** condense(**int**[] n){

StringBuffer sb = **new** StringBuffer();

**for**(**int** i=0;i<n.length;i++){

**char**[] c = (""+n[i]).toCharArray();

sb.append(*con*(c)); // 缩位

}

System.*out*.println(sb);

}

// 计算结果

**public** **static** **void** calc(**char**[][] c){

**int**[] n = **new** **int**[6];

**int** last = c[c.length-1].length ;

**for**(**int** i=0;i<6;i++){

**for**(**int** j=0;j<c.length-1;j++){

n[i] += c[j][i]; // 前几行列相加

}

}

**for**(**int** j=0;j<last;j++){

n[j] += c[c.length-1][j]; // 加上最后一行

}

*condense*(n); // 缩位处理,并输出结果

}

// 进入主要操作函数

**public** **static** **void** f(String s){

**int** t = (s.length()+5)/6; // 分为几组

**char**[][] c = **new** **char**[t][];

**for**(**int** i=0;i<t;i++){ // 得到分组后的数据

**if**(s.length()>6){

c[i] = s.substring(0, 6).toCharArray();

s = s.substring(6);

}**else**{

c[i] = s.substring(0, s.length()).toCharArray();

**break**;

}

}

*calc*(c); // 计算结果

}

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

Scanner scan = **new** Scanner(System.*in*); // 输入整数

Scanner scan2 = **new** Scanner(System.*in*); // 输入字符串

System.*out*.println("输入整数n（<100）表示下边有多少输入行:");

**int** n = scan.nextInt();

String[] ss = **new** String[n];

**int** k=0;

**while**(k<n){

ss[k++] = scan2.nextLine();

}

**for**(**int** i=0;i<n;i++){

*f*(ss[i]);

}

}

}

运行结果:

输入整数n（<100）表示下边有多少输入行:

5

zhangfeng

wangximing

jiujingfazi

woaibeijingtiananmen

haohaoxuexi

772243

344836

297332

716652

875843

17. 转方阵

/\*

对一个方阵转置，就是把原来的行号变列号，原来的列号变行号

例如，如下的方阵：

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

转置后变为：

1 5 9 13

2 6 10 14

3 7 11 15

4 8 12 16

但，如果是对该方阵顺时针旋转（不是转置），却是如下结果：

13 9 5 1

14 10 6 2

15 11 7 3

16 12 8 4

下面的代码实现的功能就是要把一个方阵顺时针旋转。

\*/

**public** **class** Demo03 {

// 矩阵顺时针旋转

**public** **static** **void** rotation(**int**[][] n,**int** [][] m,**int** i,**int** j){

**int** t = j; // 标记最后一行的位置

**if**(i>=n.length) **return**;

**for**(**int** k=0;k<n.length;k++){

m[i][k] = n[j--][i]; // 解决一行

}

*rotation*(n,m,++i,t); // 递归解决下一行

}

// 输出矩阵

**public** **static** **void** print(**int**[][] t){

**for**(**int**[] x: t){

**for**(**int** y:x){

System.*out*.print(y+"\t");

}

System.*out*.println();

}

}

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

**int**[][] n = {

{1 ,2 ,3 ,4 },

{5 ,6 ,7 ,8 },

{9 ,10,11,12},

{13,14,15,16}

};

*print*(n); // 显示原矩阵

**int** len = n.length;

**int**[][] m = **new** **int**[len][len]; // 目标矩阵

*rotation*(n,m,0,len-1); // 矩阵顺时针旋转

System.*out*.println("顺时针旋转结果:");

*print*(m); // 显示目标矩阵

}

}

运行结果：

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

顺时针旋转结果:

13 9 5 1

14 10 6 2

15 11 7 3

16 12 8 4

18. 古堡算式

/\*

福尔摩斯到某古堡探险，看到门上写着一个奇怪的算式：

ABCDE \* ? = EDCBA

他对华生说：“ABCDE应该代表不同的数字，问号也代表某个数字！”

华生：“我猜也是！”

于是，两人沉默了好久，还是没有算出合适的结果来。

请你利用计算机的优势，找到破解的答案。

把 ABCDE 所代表的数字写出来。

答案写在“解答.txt”中，不要写在这里！

\*/

**public** **class** TDemo02\_two {

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

**for**(**int** i=10000;i<100000;i++){

**int** a = i/10000;

**int** b = i%10000/1000;

**int** c = i%10000%1000/100;

**int** d = i%10000%1000%100/10;

**int** e = i%10;

**if**(a==b||a==c||a==d||a==e||b==c||b==d||b==e||c==d||c==e||d==e){

**continue**;

}

**int** y = e\*10000+d\*1000+c\*100+b\*10+a;

**if**(y%i==0){

System.*out*.println(i+"\*"+y/i+"="+y);

}

}

}

}

运行结果：

21978\*4=87912

19. 微生物增殖

/\*

假设有两种微生物 X 和 Y

X出生后每隔3分钟分裂一次（数目加倍），Y出生后每隔2分钟分裂一次（数目加倍）。

一个新出生的X，半分钟之后吃掉1个Y，并且，从此开始，每隔1分钟吃1个Y。

现在已知有新出生的 X=10, Y=89，求60分钟后Y的数目。

如果X=10，Y=90 呢？

本题的要求就是写出这两种初始条件下，60分钟后Y的数目。

题目的结果令你震惊吗？这不是简单的数字游戏！真实的生物圈有着同样脆弱的性质！也许因为你消灭的那只 Y 就是最终导致 Y 种群灭绝的最后一根稻草！

请忍住悲伤，把答案写在“解答.txt”中，不要写在这里！

\*/

**public** **class** Demo01 {

**public** **static** **void** microAdd(**long** x,**long** y,**int** m){

**for**(**int** i=1;i<=m;i++){

y -= x; // 吃掉x个y

**if**(i%3==0) x \*= 2; // 每隔3分钟x分裂一次

**if**(i%2==0) y \*= 2; // 每隔2分钟y分裂一次

}

System.*out*.println(y);

}

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

*microAdd*(10,89,60);

*microAdd*(10,90,60);

}

}

运行结果：

-979369984

94371840

20. 输入信用卡号码

/\*

当你输入信用卡号码的时候，有没有担心输错了而造成损失呢？其实可以不必这么担心，

因为并不是一个随便的信用卡号码都是合法的，它必须通过Luhn算法来验证通过。

该校验的过程：

1、从卡号最后一位数字开始，逆向将奇数位(1、3、5等等)相加。

2、从卡号最后一位数字开始，逆向将偶数位数字，先乘以2（如果乘积为两位数，则将其减去9），再求和。

3、将奇数位总和加上偶数位总和，结果应该可以被10整除。

例如，卡号是：5432123456788881

则奇数、偶数位（用红色标出）分布：5432123456788881

奇数位和=35

偶数位乘以2（有些要减去9）的结果：1 6 2 6 1 5 7 7，求和=35。

最后35+35=70 可以被10整除，认定校验通过。

请编写一个程序，从键盘输入卡号，然后判断是否校验通过。通过显示：“成功”，否则显示“失败”。

比如，用户输入：356827027232780

程序输出：成功

\*/

**import** java.util.Scanner;

**public** **class** Demo10 {

**public** **static** **int** calc(String s){

**int** odd = 0;

**int** even = 0;

**int** t = 0;

**char**[] c = s.toCharArray();

**if**(c.length%2==0){ // 如果位数为偶数个,则第一个数从偶数开始算起

**for**(**int** i=0;i<c.length;i++){

t = c[i]-'0';

**if**(i%2!=0){

odd += t;

}**else**{ // 第一个数加入到偶数

**if**(t\*2>=10){

even += t\*2-9;

}**else**{

even += t\*2;

}

}

}

}**else**{ // 如果位数为奇数个,则第一个数从奇数开始算起

**for**(**int** i=0;i<c.length;i++){

t = c[i]-'0';

**if**(i%2==0){ // 第一个数加入到奇数

odd += t;

}**else**{

**if**(t\*2>=10){

even += t\*2-9;

}**else**{

even += t\*2;

}

}

}

}

**return** odd+even; // 返回奇数位总和加上偶数位总和

}

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

Scanner scan = **new** Scanner(System.*in*);

System.*out*.print("输入卡号:");

String s = scan.nextLine();

**if**(*calc*(s)%10==0){ // 结果可以被10整除

System.*out*.println("成功");

}**else**{

System.*out*.println("失败");

}

}

}

运行结果：

输入卡号:5432123456788881

成功

21. 输入日期

/\*

从键盘输入一个日期，格式为yyyy-M-d

要求计算该日期与1949年10月1日距离多少天

例如：

用户输入了：1949-10-2

程序输出：1

用户输入了：1949-11-1

程序输出：31

\*/

**import** java.util.Arrays;

**import** java.util.Scanner;

**public** **class** Demo09 {

**static** **int**[][] *days* = { // 定义平年days[0]和闰年days[1]

{0,31,28,31,30,31,30,31,31,30,31,30,31},

{0,31,29,31,30,31,30,31,31,30,31,30,31}};

// 字符串转换 整数

**public** **static** **int**[] conversion(String s) {

String[] ss = s.split("\\-");

**int**[] t = **new** **int**[3];

t[0] = Integer.*parseInt*(ss[0]);

t[1] = Integer.*parseInt*(ss[1]);

t[2] = Integer.*parseInt*(ss[2]);

**return** t;

}

// 计算任意一年内的总天数 (是365还是366)

**public** **static** **int** calcDay(**int** y,**int** m,**int** d){

**int** sum = d; // 每一年的总天数 (初始化为最后一个月的天数)

**int** i = 0; // 平年下标为 0

**if**((y%4==0&&y%100!=0)||(y%400==0)) i = 1; // 闰年下标为 1

**for**(**int** j=0;j<m;j++){

sum += *days*[i][j];

}

**return** sum;

}

// 判断是否交换输入的两个日期 (日期小者放前)

**public** **static** **void** isSwap(**int**[] o,**int**[] n){

**if**(o[0]>n[0]){ // 日期内容交换

**for**(**int** i=0;i<o.length;i++){

**int** t = o[i];

o[i] = n[i];

n[i] = t;

}

}

}

// 输出

**public** **static** String p(**int**[] s){

StringBuffer sb = **new** StringBuffer();

**for**(**int** i=0;i<s.length-1;i++){

sb.append(s[i]+"-");

}

sb.append(s[s.length-1]);

**return** sb.toString();

}

// 计算总天数

**public** **static** **int** calc(**int**[] o, **int**[] n) {

**int** sum;

**if**(n[0]==o[0]){ // 如果年份相等,侧直接计算天数

sum = *calcDay*(n[0],n[1],n[2])-*calcDay*(o[0],o[1],o[2]); // 大的日期天数-小的日期天数

}**else**{

sum = *calcDay*(o[0],12,31)-*calcDay*(o[0],o[1],o[2]); // 得到第一年日期到下一年开始的天数

**for**(**int** i=o[0]+1;i<n[0];i++){ // 中间的每年天数 (不包含第一年和最后一年)

sum += *calcDay*(i,12,31);

}

sum += *calcDay*(n[0],n[1],n[2]); // 加上最后一年的天数

}

**return** sum;

}

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

// int[] o = conversion("1949-10-2");

Scanner scan = **new** Scanner(System.*in*);

System.*out*.print("输入日期格式为:(1949-10-2)\n输入第一个日期:");

**int**[] o = *conversion*(scan.nextLine());

System.*out*.print("输入第二个日期:");

**int**[] n = *conversion*(scan.nextLine());

**int** sum = 0; // 总和

*isSwap*(o,n); // 如果o日期小于n日期,侧交换

sum = *calc*(o, n); // 计算总天数

System.*out*.println(*p*(o)+" 到 "+*p*(n)+"\n距离"+sum+"天");

}

}

运行结果：

输入日期格式为:(1949-10-2)

输入第一个日期:1949-10-2

输入第二个日期:1949-11-2

1949-10-2 到 1949-11-2

距离31天

22. 顺时针螺旋填入

/\*

从键盘输入一个整数（1~20）

则以该数字为矩阵的大小，把1,2,3…n\*n 的数字按照顺时针螺旋的形式填入其中。例如：

输入数字2，则程序输出：

1 2

4 3

输入数字3，则程序输出：

1 2 3

8 9 4

7 6 5

输入数字4， 则程序输出：

1 2 3 4

12 13 14 5

11 16 15 6

10 9 8 7

\*/

**import** java.util.Scanner;

**public** **class** Demo08 {

**public** **static** **void** show(**int**[][] m) {

**for**(**int**[] x:m){

**for**(**int** y:x){

System.*out*.print(y+"\t");

}

System.*out*.println("");

}

}

// 顺时针螺旋

**public** **static** **int**[][] helix(**int** n) {

**int**[][] m = **new** **int**[n][n]; // 产生空矩阵

**int** t = n\*n; // 填充矩阵从1到n\*n个数

**int** i = 0; // 填充数字的位置(行下标)

**int** j = 0; // 填充数字的位置(列下标)

**int** start = 0; // 每填充一圈时用的边界(左上)

**int** end = n; // 每填充一圈时用的边界(右下)

**int** x = 0; // 填充左侧一竖列数字时用的起始位置(行下标)

**int** y = 0; // 填充左侧一竖列数字时用的起始位置(列下标)

**int** count = 0; // 标记那段程序执行,那段程序不执行

**for**(**int** s=1;s<=t;s++){

**if**(i==start&&j<end){ // 如果是第一行

m[i][j++] = s; // 填充第一行

}**else** **if**(i<end-1&&count<1){ // count==0时处理

j = end-1; // 列设置为右边界

i++; // 行下标 下移

}

**if**(i>start&&i<end){

**if**(j>start){

**if**(i==end-1){ // 到下边界了

m[i][j--] = s; // 填充下边界

}**else**{

m[i][j] = s; // 填充右侧一竖列数字

}

}**else** **if**(count==0){ // 只当count==0 时执行一次

m[i][j] = s; // 填充到了左下角的元素

i--; // 开始从下向上填充左侧元素(行坐标上移)

x = i; // 用新的下标x来填充左侧 (行下标)

y = j; // 用新的下标y来填充左侧 (列下标)

count++;

}**else** **if**(x>start&&y<end){

m[x--][y] = s; // 填充左侧

}**else** **if**(x==start){ // 如果外圈填充完毕

s--; // 接下来填充的数字放到下一的内圈里(抵消循环的s++)

start++; // 左上角边界向内收缩 1

end = --n; // 右下角边界向内收缩 1

i = start; // 按边界的位置开始填充内圈(行下标)

j = start; // 按边界的位置开始填充内圈(列下标)

x = i; // 填充左侧一竖列数字时用的起始位置(行下标)

y = j; // 填充左侧一竖列数字时用的起始位置(列下标)

count = 0; // 恢复执行程序段的标记

}

}

}

**return** m; // 返回矩阵

}

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

Scanner scan = **new** Scanner(System.*in*);

System.*out*.print("输入一个整数:");

**int** n = scan.nextInt(); // 输入 n

**int**[][] m = *helix*(n); // 矩阵大小为n\*n

*show*(m); // 输出矩阵

}

}

运行结果：

输入一个整数:4

1 2 3 4

12 13 14 5

11 16 15 6

10 9 8 7

自己写的,用到的控制变量比较多,所以程序读起来容易混乱!

**整体思路**: (外圈实现:

1. 从(**左->右**)填充第一行

 2.从(**上->下**)填充右侧一列

 3.从(**右->左**)填充最后一行

4.从(**下->上**)填充左侧一列

只要最外圈能做出来,内圈同理,调整变量就可以了)

**方法二:**

**public** **class** Demo08\_two {

**public** **static** **void** show(**int**[][] m) {

**for** (**int**[] x : m) {

**for** (**int** y : x) {

System.*out*.print(y + "\t");

}

System.*out*.println("");

}

}

**public** **static** **void** helix(**int** n, **int** b, **int**[][] a) {

**int** i;

**int** j;

**int** k;

**for** (i = 0; i < n / 2; i++) {

**for** (j = i; j < n - i; j++)

/\* 四个循环按不同的方向进行 \*/

a[i][j] = ++b;

**for** (k = i + 1, j--; k < n - i; k++)

a[k][j] = ++b;

**for** (j = --k, j--; j >= i; j--)

a[k][j] = ++b;

**for** (k--; k > i; k--)

a[k][i] = ++b;

}

**if** (n % 2 != 0) /\* 如果是单数的话，要加上最大的那个数放在中间 \*/

a[i][i] = ++b;

}

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

Scanner scan = **new** Scanner(System.*in*);

**int** i, j, k, n, b = 0;

System.*out*.print("输入一个整数:");

n = scan.nextInt();

**int**[][] a = **new** **int**[n][n];

*helix*(n, b, a);

*show*(a);

}

}

运行结果：

输入一个整数:4

1 2 3 4

12 13 14 5

11 16 15 6

10 9 8 7

23. Playfair密码

/\*

\* 一种Playfair密码变种加密方法如下：首先选择一个密钥单词（称为pair）（字母不重复，且都为小写字母），

\* 然后与字母表中其他字母一起填入至一个5x5的方阵中，填入方法如下：

1.首先按行填入密钥串。

2.紧接其后，按字母序按行填入不在密钥串中的字母。

3.由于方阵中只有25个位置，最后剩下的那个字母则不需变换。

如果密钥为youandme，则该方阵如下：

y o u a n

d m e b c

f g h i j

k l p q r

s t v w x

在加密一对字母时，如am，在方阵中找到以这两个字母为顶点的矩形（红色字体）：

y o u a n

d m e b c

f g h i j

k l p q r

s t v w x

这对字母的加密字母为该矩形的另一对顶点，如本例中为ob。

请设计程序，使用上述方法对输入串进行加密，并输出加密后的串。

另外有如下规定：

1、一对一对取字母，如果最后只剩下一个字母，则不变换，直接放入加密串中；

2、如果一对字母中的两个字母相同，则不变换，直接放入加密串中；

3、如果一对字母中有一个字母不在正方形中"如z"，则不变换，直接放入加密串中；

4、如果字母对出现在方阵中的同一行或同一列，如df或hi，则只需简单对调这两个字母，即变换为fd或ih；

5、如果在正方形中能够找到以字母对为顶点的矩形，假如字母对为am，则该矩形的另一对顶点字母中，

与a同行的字母应在前面，在上例中应是ob；同样若待变换的字母对为ta，则变换后的字母对应为wo；

6、本程序中输入串均为小写字母，并不含标点、空格或其它字符。

解密方法与加密相同，即对加密后的字符串再加密，将得到原始串。

要求输入形式如下：

从控制台输入两行字符串，第一行为密钥单词（长度小于等于25），第二行为待加密字符串（长度小于等于50），

两行字符串末尾都有一个回车换行符，并且两行字符串均为小写字母，不含其它字符。

在标准输出上输出加密后的字符串。

例如，若输入：

youandme

welcometohangzhou

则表示输入的密钥单词为youandme，形成的正方形如上所示；待加密字符串为welcometohangzhou。

在正方形中可以找到以第一对字母we为顶点的矩形，对应另一对顶点字母为vb，因此加密后为vb，

同理可找到与字母对lc,et,oh,ho对应的顶点字母对。而字母对om位于上述正方形中的同一列，

所以直接以颠倒这两个字母来加密，即为mo，字母对an同理。字母对gz中的z不在上述正方形中，

因此原样放到加密串中。最后剩一个字母u也原样输出。

因此输出的结果为：

vbrmmomvugnagzguu

vbrmmomvugnagzguu

vbrmmomvugnagzguu

要求考生把所有类写在一个文件中。调试好后，存入与考生文件夹下对应题号的“解答.txt”中即可。

相关的工程文件不要拷入。请不要使用package语句。

另外，源程序中只能出现JDK1.5中允许的语法或调用。不能使用1.6或更高版本。

\*/

**import** java.util.Scanner;

**public** **class** Demo09 {

**static** **char** *a* = 'a';

// 返回不包含在key里的元素

**public** **static** **char** contains(String k){

**if**(k.indexOf(*a*)>0){ // 如果 k 包含 c 则 a++ 再进行测试

*a*++;

*contains*(k);

}

**return** *a*; // 返回不包含在key里的元素

}

// 显示矩阵

**public** **static** **void** show(**char**[][] m){

**for**(**int** i=0;i<5;i++){

**for**(**int** j=0;j<5;j++){

System.*out*.print(" "+m[i][j]);

}

System.*out*.println();

}

}

// 完成矩阵添加元素

**public** **static** **void** fillmatrix(String key, **char**[][] m) {

**int** len = 0;

**for**(**int** i=0;i<5;i++){

**for**(**int** j=0;j<5;j++){

**if**(len<key.length()){

m[i][j] = key.charAt(len);

len++;

}**else**{ // 添加 (a~z在key里不包含)的元素

m[i][j] = *contains*(key);

*a*++;

}

}

}

}

// 待加密的字符串分对(2个字符一对)

**public** **static** String[] pair(String str){

String[] ss = **new** String[(str.length()+1)/2];

StringBuffer sb = **new** StringBuffer(str);

**int** i = 0;

**while**(sb.length()!=0){

**if**(sb.length()<=1){

ss[i] = sb.toString();

**break**;

}**else**{

ss[i++] = sb.substring(0,2);

sb.delete(0, 2);

}

}

**return** ss;

}

**public** **static** String find(String ss,**char**[][] m){

**char**[] c = ss.toCharArray();

**if**(c.length==1) **return** ss; // 如果是一个字符,则直接返回

**int** count = 0; // 统计c里的两个字符中否都存在

**int**[] f = **new** **int**[4]; // 保存下标

StringBuffer sb = **new** StringBuffer();

**for**(**int** i=0;i<m.length;i++){

**for**(**int** j=0;j<m[i].length;j++){

**if**(m[i][j]==c[0]){

f[0] = i; // f[0,1]保存第一个字符的位置

f[1] = j;

count++; // 存在加1

}**else** **if**(m[i][j]==c[1]){

f[2] = i; // f[0,1]保存第二个字符的位置

f[3] = j;

count++; // 存在加1

}

}

}

**if**(count!=2) **return** ss; //count!=2说明有一个字符在矩阵里不存在

sb.append(m[f[0]][f[3]]); // 添加加密后的第一个字符

sb.append(m[f[2]][f[1]]); // 添加加密后的第二个字符

**return** sb.toString();

}

// 加密

**public** **static** String encry(String str,**char**[][] m){

StringBuffer sb = **new** StringBuffer();

String[] ss = *pair*(str);

**for**(**int** i=0;i<ss.length;i++){

sb.append(*find*(ss[i],m));

}

**return** sb.toString();

}

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

Scanner scan = **new** Scanner(System.*in*);

System.*out*.print("输入密钥单词:");

String key = scan.nextLine();

System.*out*.print("输入待加密字符:");

String str = scan.nextLine();

**char**[][] m = **new** **char**[5][5];

*fillmatrix*(key, m); // 完成矩阵添加元素

*show*(m); // 显示矩阵

String e = *encry*(str,m); // 加密

System.*out*.println("加密结果:"+e); // 输出加密后的字符串

String d = *encry*(e,m); // 解密

System.*out*.println("解密结果:"+d); // 输出加密后的字符串

}

}

运行结果：

输入密钥单词:youandme

输入待加密字符:welcometohangzhou

y o u a n

d m e b c

f g h i j

k l p q r

s t v w x

加密结果:vbrmommvugnagzguu

解密结果:welcometohangzhou

24. 金额组合

/\*

某财务部门结账时发现总金额不对头。很可能是从明细上漏掉了某1笔或几笔。

如果已知明细账目清单，能通过编程找到漏掉的是哪1笔或几笔吗？

如果有多种可能，则输出所有可能的情况。

我们规定：用户输入的第一行是：有错的总金额。

接下来是一个整数n，表示下面将要输入的明细账目的条数。

再接下来是n行整数，分别表示每笔账目的金额。

要求程序输出：所有可能漏掉的金额组合。每个情况1行。金额按照从小到大排列，中间用空格分开。

比如：

用户输入：

6

5

3

2

4

3

1

表明：有错的总金额是6；明细共有5笔。

此时，程序应该输出：

1 3 3

1 2 4

3 4

为了方便，不妨假设所有的金额都是整数；每笔金额不超过1000，金额的明细条数不超过100。

\*/

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Iterator;

**import** java.util.LinkedHashSet;

**import** java.util.List;

**import** java.util.Scanner;

**import** java.util.Set;

**public** **class** Demo08 {

**static** Set<List<Integer>> *sets* = **new** LinkedHashSet<List<Integer>>(); // 保存记录

// 得到漏掉的总金额

**public** **static** **int** sum(**int**[] d){

**int** leak = 0;

**for**(**int** i=0;i<d.length;i++){

leak += d[i];

}

**return** leak;

}

// 修改b元素+1进位

**public** **static** **void** modify(**int**[] b){

b[b.length-1]++;

**for**(**int** i=b.length-1;i>0;i--){

**if**(b[i]>1){

b[i] = 0;

b[i-1]++;

}

}

}

// 检测b是否全111111...

**public** **static** **boolean** check(**int**[] b){

**boolean** flag = **true**;

**for**(**int** i=0;i<b.length;i++){

**if**(b[i]==0){

flag = **false**;

**break**;

}

}

**return** flag;

}

**public** **static** **void** calc(**int** le,**int**[] d,**int**[] b){

**if**(*check*(b)){

**return**;

}

**int** sum = 0;

**for**(**int** i=d.length-1;i>=0;i--){

**if**(b[i]!=0){

sum += d[i];

}

}

**if**(sum==le){

List<Integer> lis = **new** ArrayList();

**for**(**int** i=0;i<d.length;i++){

**if**(b[i]!=0){

lis.add(d[i]);

}

}

*sets*.add(lis);

//lis.clear();

}

*modify*(b);

*calc*(le,d,b);

}

// 输出结果

**public** **static** **void** print(){

Iterator<List<Integer>> iter = *sets*.iterator();

**while**(iter.hasNext()){

Iterator<Integer> iter2 = iter.next().iterator();

**while**(iter2.hasNext()){

System.*out*.print(iter2.next()+" ");

}

System.*out*.println();

}

}

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

Scanner scan = **new** Scanner(System.*in*);

**int** err = scan.nextInt();

**int** n = scan.nextInt();

**int**[] d = **new** **int**[n];

**for**(**int** i=0;i<n;i++){

d[i] = scan.nextInt();

}

Arrays.*sort*(d);

**int** le = *sum*(d) - err; // 得到漏掉的总金额

// 初始为全零 作为标记,记录数字是否使用 0为未使用 1为已经使用

**int**[] b = **new** **int**[d.length];

*calc*(le,d,b);

System.*out*.println("输出结果:");

*print*();

}

}

运行结果：

6

5

3

2

4

3

1

输出结果:

3 4

1 3 3

1 2 4

25. 拼出漂亮的表格

/\*

\* 在中文Windows环境下，控制台窗口中也可以用特殊符号拼出漂亮的表格来。

比如：

┌─┬─┐

│ │ │

├─┼─┤

│ │ │

└─┴─┘

其实，它是由如下的符号拼接的：

左上 = ┌

上 = ┬

右上 = ┐

左 = ├

中心 = ┼

右 = ┤

左下= └

下 = ┴

右下 = ┘

垂直 = │

水平 = ─

本题目要求编写一个程序，根据用户输入的行、列数画出相应的表格来。

例如用户输入：

3 2

则程序输出：

┌─┬─┐

│ │ │

├─┼─┤

│ │ │

├─┼─┤

│ │ │

└─┴─┘

用户输入：

2 3

则程序输出：

┌─┬─┬─┐

│ │ │ │

├─┼─┼─┤

│ │ │ │

└─┴─┴─┘

要求考生把所有类写在一个文件中。调试好后，存入与考生文件夹下对应题号的“解答.txt”中即可。相关的工程文件不要拷入。请不要使用package语句。

另外，源程序中只能出现JDK1.5中允许的语法或调用。不能使用1.6或更高版本。

\*/

**import** java.util.Scanner;

**public** **class** Demo07 {

// 输出中间全部都是竖杠的行

**public** **static** **void** middle(**int** j){

**for**(**int** i=0;i<j;i++){

System.*out*.print("│ ");

}

System.*out*.println("│");

}

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

Scanner scan = **new** Scanner(System.*in*);

System.*out*.println("请输出两个数,行和列:");

**int** row = scan.nextInt();

**int** col = scan.nextInt();

**for**(**int** i=0;i<row;i++){

**for**(**int** j=0;j<=col;j++){

**if**(i==0){ // 输出第一行

**if**(j==0){

System.*out*.print("┌─");

}**else** **if**(j>0&&j<col){

System.*out*.print("┬─");

}**else**{

System.*out*.println("┐");

*middle*(col);

}

}**else** **if**(i==row){ // 输出最后一行

**if**(j==0){

System.*out*.print("└─");

}**else** **if**(j>0&&j<col){

System.*out*.print("┴─");

}**else**{

System.*out*.println("┘");

}

}**else**{ // 输出中间行

**if**(j==0){

System.*out*.print("├─");

}**else** **if**(j>0&&j<col){

System.*out*.print("┼─");

}**else**{

System.*out*.println("┤");

*middle*(col);

}

}

}

}

}

}

运行结果：

请输出两个数,行和列 ,例: 3 4

3 4

┌─┬─┬─┬─┐

│ │ │ │ │

├─┼─┼─┼─┤

│ │ │ │ │

├─┼─┼─┼─┤

│ │ │ │ │

└─┴─┴─┴─┘

26. 迷宫问题

/\*

迷宫问题

对于走迷宫，人们提出过很多计算机上的解法。深度优先搜索、广度优先搜索是使用最广的方法。生活中，人们更愿意使用“紧贴墙壁，靠右行走”的简单规则。

下面的代码则采用了另一种不同的解法。它把走迷宫的过程比做“染色过程”。

假设入口点被染为红色，它的颜色会“传染”给与它相邻的可走的单元。

这个过程不断进行下去，如果最终出口点被染色，则迷宫有解。

仔细分析代码中的逻辑，填充缺少的部分。把填空的答案（仅填空处的答案，不包括题面）存入考生文件夹下对应题号的“解答.txt”中即可。

\*/

**import** java.util.HashSet;

**import** java.util.Iterator;

**import** java.util.Set;

**class** Cell{ // 单元格

**int** row; // 哪行

**int** col; // 哪列

Cell from; // 开始点

**public** Cell(**int** row,**int** col,Cell from){

**this**.row = row;

**this**.col = col;

**this**.from = from;

}

}

**public** **class** T04 {

**static** **char**[][] *maze* = {

{'#','#','#','#','B','#','#','#','#','#','#','#'},

{'#','#','#','#','.','.','.','.','#','#','#','#'},

{'#','#','#','#','.','#','#','#','#','.','.','#'},

{'#','.','.','.','.','#','#','#','#','#','.','#'},

{'#','.','#','#','#','#','#','.','#','#','.','#'},

{'#','.','#','#','#','#','#','.','#','#','.','#'},

{'#','.','#','#','.','.','.','.','.','.','.','#'},

{'#','.','#','#','.','#','#','#','.','#','.','#'},

{'#','.','.','.','.','#','#','#','.','#','.','#'},

{'#','#','.','#','.','#','#','#','.','#','.','A'},

{'#','#','.','#','#','#','.','.','.','#','#','#'},

{'#','#','#','#','#','#','#','#','#','#','#','#'}

};

// 显示迷宫

**public** **static** **void** show(){

**for**(**int** i=0;i<*maze*.length;i++){

**for**(**int** j=0;j<*maze*[i].length;j++){

System.*out*.print(" "+*maze*[i][j]);

}

System.*out*.println();

}

}

// 染色

**public** **static** Cell colorCell(Set<Cell> from,Set<Cell> desc){

Iterator<Cell> iter = from.iterator();

**while**(iter.hasNext()){

Cell a = iter.next();

Cell c[] = **new** Cell[4];

c[0] = **new** Cell(a.row-1,a.col,a); // 向上走

c[1] = **new** Cell(a.row+1,a.col,a); // 向下走

c[2] = **new** Cell(a.row,a.col-1,a); // 向左走

c[3] = **new** Cell(a.row,a.col+1,a); // 向右走

**for**(**int** i=0;i<4;i++){

**if**(c[i].row<0 || c[i].row>=*maze*.length) **continue**;

**if**(c[i].col<0 || c[i].col>=*maze*[0].length) **continue**;

**char** x = *maze*[c[i].row][c[i].col]; // 取得单元格对应字符

**if**(x=='B') **return** a;

**if**(x=='.'){

*maze*[c[i].row][c[i].col] = '?'; // 染色

desc.add(c[i]);

}

}

}

**return** **null**;

}

//

**public** **static** **void** resolve(){

Set<Cell> set = **new** HashSet<Cell>();

set.add(**new** Cell(9,11,**null**));

**for**(;;){

Set<Cell> set1 = **new** HashSet<Cell>();

// 出口 a.from.from.from.....<-(set.get(0).from)==null<-入口

Cell a = *colorCell*(set,set1);

**if**(a!=**null**){ // 找到解

System.*out*.println("找到解!");

**while**(a!=**null**){ // 当前a里包含a.from 一直往前推

*maze*[a.row][a.col] = '+'; // 染色路径

a = a.from;

}

**break**;

}

**if**(set1.isEmpty()){ // 遍历所以一直到没有路走,这时 set1为空

System.*out*.println("无解!");

**break**;

}

set = set1; // 向里边

}

}

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

*show*();

*resolve*();

*show*();

}

}

运行结果：

# # # # B # # # # # # #

# # # # . . . . # # # #

# # # # . # # # # . . #

# . . . . # # # # # . #

# . # # # # # . # # . #

# . # # # # # . # # . #

# . # # . . . . . . . #

# . # # . # # # . # . #

# . . . . # # # . # . #

# # . # . # # # . # . A

# # . # # # . . . # # #

# # # # # # # # # # # #

找到解!

# # # # B # # # # # # #

# # # # + . . . # # # #

# # # # + # # # # ? ? #

# + + + + # # # # # ? #

# + # # # # # ? # # ? #

# + # # # # # ? # # ? #

# + # # + + + + + + + #

# + # # + # # # ? # + #

# + + + + # # # ? # + #

# # ? # ? # # # ? # + +

# # ? # # # ? ? ? # # #

# # # # # # # # # # # #

27. 数字黑洞

/\*

任意一个5位数，比如：34256，把它的各位数字打乱，重新排列，可以得到

一个最大的数：65432，一个最小的数23456。

求这两个数字的差，得：41976，把这个数字再次重复上述过程（如果不足5位，则前边补0）。如此往复，数字会落入某个循环圈（称为数字黑洞）。

比如，刚才的数字会落入：[82962,75933, 63954, 61974]这个循环圈。

请编写程序，找到5位数所有可能的循环圈，并输出，每个循环圈占1行。

其中5位数全都相同则循环圈为[0]，这个可以不考虑。循环圈的输出格式仿照：

[82962,75933, 63954, 61974]其中数字的先后顺序可以不考虑。

\*/

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collections;

**import** java.util.Iterator;

**import** java.util.LinkedHashSet;

**import** java.util.List;

**import** java.util.Set;

**public** **class** BlackHole {

**static** Set<List<Integer>> *sets* = **new** LinkedHashSet<List<Integer>>();// 记录每个 循环圈

**static** **int** *start* = 0; // 记录循环圈的开始位置

// 得到最大的数

**public** **static** **int** max(String s){

**char**[] c = s.toCharArray();

Arrays.*sort*(c);

StringBuffer sb = **new** StringBuffer(**new** String(c));

**return** Integer.*parseInt*(sb.reverse().toString());

}

// 得到最小的数

**public** **static** **int** min(String s){

**char**[] c = s.toCharArray();

Arrays.*sort*(c);

**return** Integer.*parseInt*(**new** String(c));

}

// 补零

**public** **static** String fillZero(String s){

**if**(s.length()<5) **return** s; // 如果够五位的话,直接返回原数

StringBuffer sb = **new** StringBuffer(s);

**int** len = 5 - sb.length(); // 需要补 len 个零

**for**(**int** i=0;i<len;i++){

sb.insert(0, "0");

}

**return** sb.toString();

}

// 查找是否有循环圈

**public** **static** **boolean** find(**int** n,List<Integer> lis){

**for**(**int** i=0;i<lis.size();i++){

**if**(n==lis.get(i)){

*start* = i; // 设置循环圈的开始

**return** **true**;

}

}

**return** **false**;

}

// 这个数字的五位是否都一样如:11111,22222,...

**public** **static** **boolean** same(**int** n){

**char**[] c = (""+n).toCharArray();

**char** t = c[0];

**for**(**int** i=1;i<c.length;i++){

**if**(t!=c[i]){

**return** **false**;

}

}

**return** **true**;

}

// 判断是否在sets元素里重复

**public** **static** **boolean** contain(List<Integer> tt){

**boolean** flag = **false**;

// 从sets头到尾扫描是是否包含tt元素组

Iterator<List<Integer>> iter = *sets*.iterator();

**while**(iter.hasNext()){

**if**(iter.next().containsAll(tt)){

flag = **true**;

}

}

**return** flag;

}

// 输出循环圈

**public** **static** **void** print(){

Iterator<List<Integer>> iter = *sets*.iterator();

**while**(iter.hasNext()){

System.*out*.println(iter.next());

}

}

// 进入黑洞测试\*\*\*\*\*\*\*\*\*

**public** **static** **void** bl(**int** n,List<Integer> lis){

String tt = *fillZero*(String.*valueOf*(n));

**int** a = *max*(tt); // 得到最大的数

**int** b = *min*(tt); // 得到最小的数

**int** c = a - b; // 得到五位数 结果

**if**(*find*(c,lis)){ // 找到循环圈

lis.add(c); // 把最后一个元素添加进去

List<Integer> temp = **new** ArrayList(); // 开辟新空间 并copy记录

temp.addAll(lis.subList(*start*, lis.size()-1));

**if**(!*contain*(temp)){ // 去重

*sets*.add(temp); // 记录一个循环圈

}

lis.clear(); // 清空

**return** ;

}

lis.add(c);

*bl*(c,lis); // 递归探测

}

// 主函数============

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

List<Integer> lis = **new** ArrayList(); // 保存记录

**for**(**int** i=10000;i<99999;i++){

**if**(!*same*(i)) //去掉不符合条件的元素:如(11111,22222,...)

*bl*(i,lis); // 进入黑洞测试

}

*print*(); // 输出循环圈

}

}

运行结果：

[0]

[74943, 62964, 71973, 83952]

[63954, 61974, 82962, 75933]

[53955, 59994]

28. 基因牛

/\*

张教授采用基因干预技术成功培养出一头母牛，三年后，这头母牛每年会生出1头母牛，

生出来的母牛三年后，又可以每年生出一头母牛。如此循环下去，请问张教授n年后有多少头母牛?

以下程序模拟了这个过程，请填写缺失的代码。

\*/

**import** java.util.ArrayList;

**import** java.util.List;

**class** Cow{

**private** **int** age;

**public** Cow afterYear(){

age++;

**return** age > 2 ? **new** Cow() : **null**; // 填空

}

**public** **static** **void** showTotalCowNum(**int** n){

List<Cow> list = **new** ArrayList<Cow>();

list.add(**new** Cow());

**for** (**int** i = 0; i < n; i++) {

**int** cowCount = list.size();

**for** (**int** j = 0; j < cowCount; j++){

Cow cow = list.get(j).afterYear();

**if** (cow != **null**){

cow.afterYear(); // 填空

list.add(cow);

}

}

}

System.*out*.println(n + "年后，共有：" + list.size());

}

}

**class** Demo05{

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

Cow.*showTotalCowNum*(13);

}

}

运行结果：

13年后，共有：233

29. 最近距离

/\*

已知平面上的若干点的位置，存入一个List中。现在需要计算所有这些点中，

距离最近的两个点间的最小距离。请补全缺失的代码。

把填空的答案（仅填空处的答案，不包括题面）存入考生文件夹下对应题号的“解答.txt”中即可。

\*/

**import** java.util.ArrayList;

**import** java.util.List;

**class** MyPoint{

**private** **double** x; // 横坐标

**private** **double** y; // 纵坐标

**public** MyPoint(**double** x, **double** y){

**this**.x = x;

**this**.y = y;

}

**public** **static** **double** distance(MyPoint p1, MyPoint p2){

**double** dx = p1.x - p2.x;

**double** dy = p1.y - p2.y;

**return** Math.*sqrt*(dx\*dx + dy\*dy);

}

/\*

lst中含有若干个点的坐标

返回其中距离最小的点的距离

\*/

**public** **static** **double** getMinDistance(List<MyPoint> lst){

**if**(lst==**null** || lst.size()<2) **return** Double.*MAX\_VALUE*;

**double** r = Double.*MAX\_VALUE*;

MyPoint p0 = lst.remove(0);

**for**(**int** i=0; i<lst.size(); i++){

MyPoint p = lst.get(i);

**double** d = MyPoint.*distance*(p0,p);

**if**(d<r) r = d; // 填空

}

**double** d2 = *getMinDistance*(lst);

**return** d2 < r ? d2 : r;

}

}

**class** Demo04{

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

List<MyPoint> list = **new** ArrayList<MyPoint>() ;

list.add(**new** MyPoint(31,4));

list.add(**new** MyPoint(1,2));

list.add(**new** MyPoint(1,1));

list.add(**new** MyPoint(1,4));

System.*out*.println(MyPoint.*getMinDistance*(list));

}

}

运行结果：

1.0

30.字符串反转

/\*

我们把“cba”称为“abc”的反转串。

求一个串的反转串的方法很多。下面就是其中的一种方法，代码十分简洁（甚至有些神秘），

请聪明的你通过给出的一点点线索补充缺少的代码。

把填空的答案（仅填空处的答案，不包括题面）存入考生文件夹下对应题号的“解答.txt”中即可。

\*/

**public** **class** Demo03 {

**public** **static** String reverseString(String s){

**if**(s.length()<2||s==**null**) **return** s;

**return** *reverseString*(s.substring(1))+s.charAt(0);

}

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

String s = "123456";

String ss = *reverseString*(s);

System.*out*.println(ss);

}

}

运行结果：

654321

31. 孪生素数

/\*

所谓孪生素数指的就是间隔为 2 的相邻素数，它们之间的距离已经近得不能再近了，

就象孪生兄弟一样。最小的孪生素数是 (3, 5)，

在 100 以内的孪生素数还有 (5, 7), (11, 13), (17, 19), (29, 31),

(41, 43), (59, 61) 和 (71, 73)，总计有 8 组。

但是随着数字的增大，孪生素数的分布变得越来越稀疏，寻找孪生素数也变得越来越困难。

那么会不会在超过某个界限之后就再也不存在孪生素数了呢？

孪生素数有无穷多对！这个猜想被称为孪生素数猜想，至今没有被严格证明。

但借助于计算机我们确实可以找到任意大数范围内的所有孪生素数对。

下面的代码求出了正整数n以内（不含n）的所有孪生素数对的个数。

比如，当n=100的时候，该方法返回8。试补全缺少的代码。

把填空的答案（仅填空处的答案，不包括题面）存入考生文件夹下对应题号的“解答.txt”中即可。

\*/

**class** Demo02{

// 判断是否是素数

**public** **static** **boolean** isPrime(**int** x){

**for**(**int** i=2; i<=x/2; i++){

**if**(x%i==0) **return** **false**;

}

**return** **true**;

}

// 是否是孪生素数

**public** **static** **int** twinPrimeNum(**int** n){

**int** sum = 0;

**for**(**int** i=2; i<n; i++){

**if**(*isPrime*(i) && *isPrime*(i+2) && (i+2)<=n) sum++;

}

**return** sum; // 返回个数

}

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

**int** n = 1000;

System.*out*.println(*twinPrimeNum*(n));

}

}

32. 圆周率

/\*

我国古代数学家对圆周率方面的研究工作，成绩是突出的。三国时期的刘徽、南北朝时期的祖冲之都在这个领域取得过辉煌战绩。

有了计算机，圆周率的计算变得十分容易了。如今，人们创造了上百种方法求π的值。其中比较常用且易于编程的是无穷级数法。

π/4 = 1 – 1/3 + 1/5 – 1/7 + 1/9 - …

是初学者特别喜欢的一个级数形式，但其缺点是收敛太慢。

π/2 = 1 + 1/3 +1/3\*2/5 + 1/3\*2/5\*3/7 + 1/3\*2/5\*3/7\*4/9 + …

是收敛很快的一个级数方法。下面的代码演示了用这种方法计算π值。请填写缺失的代码部分。

把填空的答案（仅填空处的答案，不包括题面）存入考生文件夹下对应题号的“解答.txt”中即可。

\*/

**public** **class** Demo01 {

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

**double** x = 1;

**double** y = 1;

**int** a = 1;

**int** b = 3;

**while**(y>1e-15){

y = y\*a/b;<span style="white-space:pre"> </span>// 填空

x += y;

a++;

b += 2;

}

System.out.println(x\*2);

}

}

34. 泊松分酒

/\*

泊松是法国数学家、物理学家和力学家。他一生致力科学事业，成果颇多。

有许多著名的公式定理以他的名字命名，比如概率论中著名的泊松分布。

有一次闲暇时，他提出过一个有趣的问题，后称为：“泊松分酒”。

在我国古代也提出过类似问题，遗憾的是没有进行彻底探索，其中流传较多是：“韩信走马分油”问题。

有3个容器，容量分别为12升，8升，5升。其中12升中装满油，另外两个空着。

要求你只用3个容器操作，最后使得某个容器中正好有6升油。

下面的列表是可能的操作状态记录：

12,0,0

4,8,0

4,3,5

9,3,0

9,0,3

1,8,3

1,6,5

每行3个数据，分别表示12，8，6升容器中的油量

第一行表示初始状态，第二行表示把12升倒入8升容器后的状态，第三行是8升倒入5升，...

当然，同一个题目可能有多种不同的正确操作步骤。

本题目的要求是，请你编写程序，由用户输入：各个容器的容量，开始的状态，

和要求的目标油量，程序则通过计算输出一种实现的步骤（不需要找到所有可能的方法）。

如果没有可能实现，则输出：“不可能”。

例如，用户输入：

12,8,5,12,0,0,6

用户输入的前三个数是容器容量（由大到小），接下来三个数是三个容器开始时的油量配置，

最后一个数是要求得到的油量（放在哪个容器里得到都可以）

则程序可以输出（答案不唯一，只验证操作可行性）：

12,0,0

4,8,0

4,3,5

9,3,0

9,0,3

1,8,3

1,6,5

每一行表示一个操作过程中的油量状态。

注意：

请仔细调试！您的程序只有能运行出正确结果的时候才有机会得分！

请把所有类写在同一个文件中，调试好后，存入与【考生文件夹】下对应题号的“解答.txt”中即可。

相关的工程文件不要拷入。

请不要使用package语句。

源程序中只能出现JDK1.5中允许的语法或调用。不能使用1.6或更高版本。

\*/

**import** java.util.Scanner;

**public** **class** Demo16 {

// 有6种倒酒方法,x[0]->y[0]即（"0"->"1"）代表第一个瓶子向第二个瓶子倒酒,后即同理

**static** **int**[] *x* = {0,1, 2}; // 0 为第一个瓶子，1为第二个瓶子 ，2为第三个瓶子

**static** **int**[] *y* = {1,2, 0};

**static** **int**[][] *rr* = **new** **int**[1000][3]; // 记录倒酒后每一行结果

**static** **int** *index* = 0;

**static** **int** *count* = 0;

**static** **int**[] *tt*= {0,0,0};

// 输入数据

**public** **static** **int**[] input(){

Scanner scan = **new** Scanner(System.*in*);

String [] s = scan.nextLine().split(",");

**int**[] temp = **new** **int**[s.length];

**for**(**int** i=0;i<s.length;i++){ // 字符转为数字

temp[i] = Integer.*parseInt*(s[i]);

}

**return** temp;

}

// 输出

**public** **static** **void** print(**int**[][] rr){

**for**(**int** i=0;i<*index*;i++){

**for**(**int** j:rr[i])

System.*out*.print(j+"\t");

System.*out*.println();

}

System.*out*.println("记录数:("+*index*+")");

}

// 记录步骤

**public** **static** **void** record(**int**[] cur){

*rr*[*index*][0] = cur[0];

*rr*[*index*][1] = cur[1];

*rr*[*index*][2] = cur[2];

*index*++;

}

// 判断当前的记录 是否 在以前的记录里 存在

**public** **static** **boolean** curExist(**int**[] cur){

**for**(**int** i=0;i<*index*;i++){

**if**(*rr*[i][0]==cur[0] && *rr*[i][1]==cur[1]

&& *rr*[i][2]==cur[2]){

**return** **true**;

}

}

**return** **false**;

}

// 倒酒

**public** **static** **void** pour(**int**[] v,**int**[] cur,**int** i){

*count*++; // 统计倒酒的次数, 若倒酒次数超过1000次,则识为"不可能"

**int** r = v[*y*[i]] - cur[*y*[i]]; // 计算 y瓶中还可以装入多少酒，拿y瓶的总容量-y瓶当前的酒

**if**(cur[*x*[i]] > r){ // x > y 时

cur[*y*[i]] = v[*y*[i]]; // y = 满

cur[*x*[i]] -= r; // x = x - r

}**else**{ // x <= y

cur[*y*[i]] += cur[*x*[i]]; // y = y + x

cur[*x*[i]] = 0; // x = x - r

}

}

// 求解

**public** **static** **void** f(**int**[] v,**int**[] cur,**int** m){

**if**(m>v[0]){

System.*out*.println("要求得到的油量"+m+"大于最大容器"+v[0]+",所以\n不可能");

**return**;

}

**boolean** flag = **true**;

*record*(cur);

**while**(flag){

**if**(*count*>1000){

System.*out*.println("倒酒次数超过1000次,所以\n不可能");

**return**;

}

**for**(**int** i=0; i<3; i++){ // 6种倒酒方法

// 找到解，退出

**if**(cur[0]==m || cur[1]==m || cur[2]==m){

*print*(*rr*); // 找到解,输出记录

flag = **false**;

**break**;

}

// 如果 x 瓶中为空，则跳过， 执行下一轮倒酒

**if**(cur[*x*[i]] == 0){

**continue**;

}

*pour*(v,cur,i);// 倒酒

// 记录步骤

**if**(*curExist*(cur)){

cur[0] = *rr*[*index*-1][0]; // 还原为上次倒酒的值

cur[1] = *rr*[*index*-1][1];

cur[2] = *rr*[*index*-1][2];

//--index;

**continue**;

}**else**{

*record*(cur);

}

}

}

}

// 主函数

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

System.*out*.println("输入数据（格式为7个数字用\",\"号隔开,例：)12,8,5,12,0,0,6");

**int**[] t = *input*(); // 输入数据

**int**[] v = {t[0],t[1],t[2]}; // 每个容器的最大容量 v

**int**[] cur = {t[3],t[4],t[5]}; // 容器的开始的状态 init

**int** m = t[6]; // 要求得到的油量 r

*f*(v,cur,m);

}

}

运行结果：

输入数据（格式为7个数字用","号隔开,例：)12,8,5,12,0,0,6

12,8,5,12,0,0,6

12 0 0

4 8 0

4 3 5

9 3 0

9 0 3

1 8 3

1 6 5

记录数:(7)

35. 矩形的关系。

/\*

在编写图形界面软件的时候，经常会遇到处理两个矩形的关系。

如图【1.jpg】所示，矩形的交集指的是：两个矩形重叠区的矩形，当然也可能不存在（参看【2.jpg】）。

两个矩形的并集指的是：能包含这两个矩形的最小矩形，它一定是存在的。

本题目的要求就是：由用户输入两个矩形的坐标，程序输出它们的交集和并集矩形。

矩形坐标的输入格式是输入两个对角点坐标，注意，不保证是哪个对角，

也不保证顺序（你可以体会一下，在桌面上拖动鼠标拉矩形，4个方向都可以的）。

输入数据格式：

x1,y1,x2,y2

x1,y1,x2,y2

数据共两行，每行表示一个矩形。每行是两个点的坐标。x坐标在左，y坐标在右。

坐标系统是：屏幕左上角为(0,0)，x坐标水平向右增大；y坐标垂直向下增大。

要求程序输出格式：

x1,y1,长度,高度

x1,y1,长度,高度

也是两行数据，分别表示交集和并集。如果交集不存在，则输出“不存在”

前边两项是左上角的坐标。后边是矩形的长度和高度。

例如，用户输入：

100,220,300,100

150,150,300,300

则程序输出：

150,150,150,70

100,100,200,200

例如，用户输入：

10,10,20,20

30,30,40,40

则程序输出：

不存在

10,10,30,30

\*/

**import** java.awt.Rectangle;

**import** java.util.Scanner;

**public** **class** Demo13\_Rectang {

**public** **static** Rectangle getRec(Rectangle[] rec){

Scanner scan = **new** Scanner(System.*in*);

String s = scan.nextLine();

String[] ss = s.split(",");

**int** x1 = Integer.*parseInt*(ss[0]);

**int** y1 = Integer.*parseInt*(ss[1]);

**int** x2 = Integer.*parseInt*(ss[2]);

**int** y2 = Integer.*parseInt*(ss[3]);

// 如果（x1,y1）（x2,y2）分别在（左上，右下，右上，左下）时

**return** **new** Rectangle(Math.*min*(x1, x2),Math.*min*(y1, y2),

Math.*abs*(x2-x1),Math.*abs*(y2-y1));

}

**public** **static** **void** op(Rectangle[] rec){

Rectangle r,rr;

**if**(rec[0].intersects(rec[1])){

r = rec[0].intersection(rec[1]); // 交集

System.*out*.println(r.x+","+r.y+","+r.width+","+r.height);

}**else**{

System.*out*.println("不存在");

}

rr = rec[0].union(rec[1]); // 并集

System.*out*.println(rr.x+","+rr.y+","+rr.width+","+rr.height);

}

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

Rectangle rec[] = **new** Rectangle[2];

rec[0] = *getRec*(rec); // 第一个矩形

rec[1] = *getRec*(rec); // 第二个矩形

*op*(rec); // 输出 交集 和 并集

}

}

运行结果1：

100,220,300,100

150,150,300,300

150,150,150,70

100,100,200,200

运行结果2：

10,10,20,20

30,30,40,40

不存在

10,10,30,30

36. 测量到的工程数据

/\*

[12,127,85,66,27,34,15,344,156,344,29,47,....]

这是某设备测量到的工程数据。

因工程要求，需要找出最大的5个值。

一般的想法是对它排序，输出前5个。但当数据较多时，这样做很浪费时间。

因为对输出数据以外的数据进行排序并非工程要求，即便是要输出的5个数字，

也并不要求按大小顺序，只要找到5个就可以。

以下的代码采用了另外的思路。考虑如果手里已经抓着5个最大数，再来一个数据怎么办呢？

让它和手里的数据比，如果比哪个大，就抢占它的座位，让那个被挤出来的再自己找位子,....

请分析代码逻辑，并推测划线处的代码。

答案写在 “解答.txt” 文件中

注意：只写划线处应该填的内容，划线前后的内容不要抄写。

\*/

**import** java.util.\*;

**public** **class** Demo11\_B23 {

**public** **static** List<Integer> max5(List<Integer> lst) {

**if** (lst.size() <= 5)

**return** lst;

**int** a = lst.remove(lst.size() - 1); // 填空

List<Integer> b = *max5*(lst);

**for** (**int** i = 0; i < b.size(); i++) {

**int** t = b.get(i);

**if** (a > t) {

lst.set(i, a); // 填空

a = t;

}

}

**return** b;

}

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

List<Integer> lst = **new** Vector<Integer>();

lst.addAll(Arrays.*asList*(12, 127, 85, 66, 27, 34, 15, 344, 156, 344,

29, 47));

System.*out*.println(*max5*(lst));

}

}

运行结果：

[344, 344, 156, 127, 85]

37. 南北朝时

/\*

\* 南北朝时，我国数学家祖冲之首先把圆周率值计算到小数点后六位，比欧洲早了1100年！

\* 他采用的是称为“割圆法”的算法，实际上已经蕴含着现代微积分的思想。

如图【1.jpg】所示，圆的内接正六边形周长与圆的周长近似。

多边形的边越多，接近的越好！我们从正六边形开始割圆吧。

如图【2.jpg】所示，从圆心做弦的垂线，可把6边形分割为12边形。

该12边形的边长a'的计算方法很容易利用勾股定理给出。

之后，再分割为正24边形，....如此循环会越来越接近圆周。

之所以从正六边开始，是因为此时边长与半径相等，便于计算。取半径值为1，开始割圆吧！

以下代码描述了割圆过程。

程序先输出了标准圆周率值，紧接着输出了不断分割过程中多边形边数和所对应的圆周率逼近值。

\*/

**public** **class** Demo10\_B21

{

**public** **static** **void** main(String[] args)

{

System.*out*.println("标准 " + Math.*PI*);

**double** a = 1;

**int** n = 6;

**for**(**int** i=0; i<10; i++)

{

**double** b = Math.*sqrt*(1-(a/2)\*(a/2));

a = Math.*sqrt*((1-b)\*(1-b) + (a/2)\*(a/2));

n = n\*2; //填空

System.*out*.println(n + " " + n\*a/2 ); // 填空

}

}

}

运行结果：

标准 3.141592653589793

12 3.105828541230249

24 3.1326286132812378

48 3.1393502030468667

96 3.14103195089051

192 3.1414524722854624

384 3.141557607911858

768 3.1415838921483186

1536 3.1415904632280505

3072 3.1415921059992717

6144 3.1415925166921577

38. 数字的值返回

/\*

以下的静态方法实现了：把串s中第一个出现的数字的值返回。

如果找不到数字，返回-1

例如：

s = "abc24us43" 则返回2

s = "82445adb5" 则返回8

s = "ab" 则返回-1

\*/

**public** **class** Demo09\_FirstNum {

**public** **static** **int** getFirstNum(String s)

{

**if**(s==**null** || s.length()==0) **return** -1;

**char** c = s.charAt(0);

**if**(c>='0' && c<='9') **return** c-'0'; //填空

**return** *getFirstNum*(s.substring(1)); //填空

}

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

String s = "abc24us43";

System.*out*.println(*getFirstNum*(s));

}

}

运行结果：

2

39. 低碳生活大奖赛

/\*

某电视台举办了低碳生活大奖赛。题目的计分规则相当奇怪：

每位选手需要回答10个问题（其编号为1到10），越后面越有难度。

答对的，当前分数翻倍；答错了则扣掉与题号相同的分数（选手必须回答问题，不回答按错误处理）。

每位选手都有一个起步的分数为10分。

某获胜选手最终得分刚好是100分，如果不让你看比赛过程，

你能推断出他（她）哪个题目答对了，哪个题目答错了吗？

如果把答对的记为1，答错的记为0，则10个题目的回答情况可以用仅含有1和0的串来表示。

例如：0010110011 就是可能的情况。

你的任务是算出所有可能情况。每个答案占一行。

\*/

**public** **class** Demo08\_a {

// 判断是否符合条件

**public** **static** **void** show(**int**[] x){

**int** s = 10;

**for**(**int** i=0;i<x.length;i++){

**if**(x[i]==0){

s = s-(i+1);&nbsp;// 扣掉与题号相同的分数

}**else**{

s = s\*2;&nbsp;// 当前分数翻倍

}

}

**if**(s==100){

**for**(**int** i:x){

System.out.print(i);

}

System.out.println();

}

}

// 递归遍历每种情况

**public** **static** **void** f(**int**[] x,**int** n){

**if**(n>=x.length){

*show*(x); // 判断

**return**;

}

x[n] = 0;

*f*(x,n+1);

x[n] = 1;

*f*(x,n+1);

}

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

**int**[] x = **new** **int**[10];

*f*(x,0);

}

}

运行结果：

0010110011

0111010000

1011010000

40. 益智玩具

import java.math.BigInteger;

/\*

\* 汉诺塔（又称河内塔）问题是源于印度一个古老传说的益智玩具。

大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着64片黄金圆盘。

大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上(可以借助第三根柱子做缓冲)。

并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。

如图【1.jpg】是现代“山寨”版的该玩具。64个圆盘太多了，所以减为7个，

金刚石和黄金都以木头代替了......但道理是相同的。

据说完成大梵天的命令需要太多的移动次数，以至被认为完成之时就是世界末日！

你的任务是精确计算出到底需要移动多少次。

很明显，如果只有2个圆盘，需要移动3次。

圆盘数为3，则需要移动7次。

那么64个呢？

答案写在“解答.txt”中，不要写在这里！

\*/

**class** Demo07\_hanoi {

**public** **static** **void** hanoi(**int** n,**char** a,**char** b,**char** c){

**if**(n>0){

*hanoi*(n-1,a,c,b);

//System.out.println(a+"->"+b);

*hanoi*(n-1,c,b,a);

*s*++;

}

}

**static** **long** *s* = 0;

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

**int** n = 10;

System.*out*.println("圆盘个数\t移动次数");

**for**(**int** i=1;i<=n;i++){

*s* = 0;

*hanoi*(i,'a','b','c');

System.*out*.println(i+" ->\t"+*s*);

}

System.*out*.println("由上结果可找到规律为：\n移动次数 = 2的n次方-1（n为圆盘个数）");

System.*out*.println("64个圆盘的移动次数为：");

BigInteger bi = **new** BigInteger("2");

bi = bi.pow(64).subtract(BigInteger.ONE);

System.*out*.println(bi);

}

}

运行结果：

圆盘个数 移动次数

1 -> 1

2 -> 3

3 -> 7

4 -> 15

5 -> 31

6 -> 63

7 -> 127

8 -> 255

9 -> 511

10 -> 1023

由上结果可找到规律为：

移动次数 = 2的n次方-1（n为圆盘个数）

64个圆盘的移动次数为：

18446744073709551615

41. 拼酒量

/\*

有一群海盗（不多于20人），在船上比拼酒量。过程如下：打开一瓶酒，

所有在场的人平分喝下，有几个人倒下了。再打开一瓶酒平分，又有倒下的，

再次重复...... 直到开了第4瓶酒，坐着的已经所剩无几，海盗船长也在其中。

当第4瓶酒平分喝下后，大家都倒下了。 等船长醒来，发现海盗船搁浅了。

他在航海日志中写到：“......昨天，我正好喝了一瓶.......奉劝大家，开船不喝酒，喝酒别开船......”

请你根据这些信息，推断开始有多少人，每一轮喝下来还剩多少人。

如果有多个可能的答案，请列出所有答案，每个答案占一行。

格式是：人数,人数,...

例如,有一种可能是：20,5,4,2,0

答案写在“解答.txt”中，不要写在这里！

\*/

**public** **class** Demo06\_a {

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

**for**(**int** i=20;i>0;i--){

**for**(**int** j=i-1;j>0;j--){

**for**(**int** k=j-1;k>0;k--){

**for**(**int** m=k-1;m>0;m--){

**if**((1.0/i+1.0/j+1.0/k+1.0/m)==1)

System.*out*.println(i+","+j+","+k+","+m+","+0);

}

}

}

}

}

}

运行结果：

20,5,4,2,0

18,9,3,2,0

15,10,3,2,0

12,6,4,2,0

42. 黄金分割数

import java.math.BigDecimal;

/\*

\* 黄金分割数0.618与美学有重要的关系。舞台上报幕员所站的位置大约就是舞台宽度的0.618处，

墙上的画像一般也挂在房间高度的0.618处，甚至股票的波动据说也能找到0.618的影子....

黄金分割数是个无理数，也就是无法表示为两个整数的比值。

0.618只是它的近似值，其真值可以通过对5开方减去1再除以2来获得，

我们取它的一个较精确的近似值：0.618034

有趣的是，一些简单的数列中也会包含这个无理数，这很令数学家震惊！

1 3 4 7 11 18 29 47 .... 称为“鲁卡斯队列”。它后面的每一个项都是前边两项的和。

如果观察前后两项的比值，即：1/3,3/4,4/7,7/11,11/18 ... 会发现它越来越接近于黄金分割数！

你的任务就是计算出从哪一项开始，这个比值四舍五入后已经达到了与0.618034一致的精度。

请写出该比值。格式是：分子/分母。比如：29/47

\*/

**public** **class** Demo05\_Fibonacci {

**public** **static** **double** format(**double** d){

BigDecimal bd = **new** BigDecimal(d).setScale(6,BigDecimal.ROUND\_HALF\_UP);

**double** dd = bd.doubleValue();

**return** dd;

}

**public** **static** **void** f(**int** a,**int** b){

**double** d = *format*((**double**)a/b);

**if**(d==0.618034){

System.*out*.println(a+"/"+b+"="+d);

**return**;

}

*f*(b,a+b);

}

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

*f*(1,3);

}

}

运行结果：

1364/2207=0.618034

43. 坐标

/\*

\* 已知平面上若干个点的坐标。

需要求出在所有的组合中，4个点间平均距离的最小值（四舍五入，保留2位小数）。

比如有4个点：a,b,c,d,则平均距离是指：ab, ac, ad, bc, bd, cd 这6个距离的平均值。

每个点的坐标表示为：横坐标,纵坐标

坐标的取值范围是：1~1000

所有点的坐标记录在in.txt中，请读入该文件，然后计算。

注意：我们测试您的程序的时候，in.txt 可能会很大，比如包含上万条记录。

举例：

如果，in.txt 内的值为：

10,10

20,20

80,50

10,20

20,10

则程序应该输出：

11.38

\*/

**import** java.io.BufferedReader;

**import** java.io.FileInputStream;

**import** java.io.InputStreamReader;

**import** java.util.List;

**import** java.util.Vector;

// 代表一个点

**class** PP {

**public** **int** x;

**public** **int** y;

**public** String toString() {

**return** x + "," + y;

}

**public** PP(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

}

}

**class** RR {

**double** x1;

**double** y1;

**double** x2;

**double** y2;

**public** RR() {

}

**public** RR(**double** x1, **double** y1, **double** x2, **double** y2) {

**this**.x1 = x1;

**this**.y1 = y1;

**this**.x2 = x2;

**this**.y2 = y2;

}

**boolean** isIn(PP p) {

**return** (x1 < p.x && p.x < x2 && y1 < p.y && p.y < y2);

}

}

**public** **class** T07 {

**public** **static** **double** f(List<PP> lst, RR r) {

// lst：当前要处理的点，r:当前区域

**if** (lst.size() < 4)

**return** 10000; // 代表很远

**if** (lst.size() < 13) {

// 测试中所有13中选4的组合

**double** min = 10000;

**for** (**int** i = 0; i < lst.size(); i++)

**for** (**int** j = i + 1; j < lst.size(); j++)

**for** (**int** k = j + 1; k < lst.size(); k++)

**for** (**int** m = k + 1; m < lst.size(); m++) {

**double** d = *distance*(lst.get(i), lst.get(j),

lst.get(k), lst.get(m));

**if** (d < min)

min = d;

}

**return** min;

}

// 缩小边界

**double** x1a = r.x2;

**double** x2a = r.x1;

**double** y1a = r.y2;

**double** y2a = r.y1;

**for** (**int** i = 0; i < lst.size(); i++) {

PP p = lst.get(i);

**if** (p.x < x1a)

x1a = p.x;

**if** (p.x > x2a)

x2a = p.x;

**if** (p.y < y1a)

y1a = p.y;

**if** (p.y > y2a)

y2a = p.y;

}

r.x1 = x1a;

r.x2 = x2a;

r.y1 = y1a;

r.y2 = y2a;

// 拆分lst 为4块,重叠4个区域

RR r1 = **new** RR();

RR r2 = **new** RR();

RR r3 = **new** RR();

RR r4 = **new** RR();

r1.x1 = r.x1;

r1.y1 = r.y1;

r1.x2 = r.x1 \* 0.25 + r.x2 \* 0.75; // 0.75\*(x2-x1) + x1

r1.y2 = r.y1 \* 0.25 + r.y2 \* 0.75;

r2.x1 = r.x1 \* 0.75 + r.x2 \* 0.25; // 0.25\*(x2-x1) + x1

r2.y1 = r.y1 \* 0.75 + r.y2 \* 0.25;

r2.x2 = r.x2;

r2.y2 = r.y2;

r3.x1 = r.x1;

r3.y1 = r.y1 \* 0.75 + r.y2 \* 0.25;

r3.x2 = r.x1 \* 0.25 + r.x2 \* 0.75;

r3.y2 = r.y2;

r4.x1 = r.x1 \* 0.75 + r.x2 \* 0.25;

r4.y1 = r.y1;

r4.x2 = r.x2;

r4.y2 = r.y1 \* 0.25 + r.y2 \* 0.75;

List<PP> t1 = **new** Vector<PP>();

List<PP> t2 = **new** Vector<PP>();

List<PP> t3 = **new** Vector<PP>();

List<PP> t4 = **new** Vector<PP>();

// isIn(p)判断是否在该矩形中

**for** (**int** i = 0; i < lst.size(); i++) {

PP p = lst.get(i);

**if** (r1.isIn(p))

t1.add(p);

**if** (r2.isIn(p))

t2.add(p);

**if** (r3.isIn(p))

t3.add(p);

**if** (r4.isIn(p))

t4.add(p);

}

// 递归

**double** d1 = *f*(t1, r1);

**double** d2 = *f*(t2, r2);

**double** d3 = *f*(t3, r3);

**double** d4 = *f*(t4, r4);

**double** d = d1;

**if** (d2 < d)

d = d2;

**if** (d3 < d)

d = d3;

**if** (d4 < d)

d = d4;

**return** d;

}

// 计算 4个点间平均距离

**public** **static** **double** distance(PP a,PP b,PP c,PP d){

**double** dis =

(*distance*(a,b) + *distance*(a,c) + *distance*(a,d) +

*distance*(b,c) + *distance*(b,d) +

*distance*(c,d)) / 6.0;

**return** dis;

}

// 求两个点的距离

**public** **static** **double** distance(PP a, PP b){

**double** dx = a.x - b.x;

**double** dy = a.y - b.y;

**return** Math.*sqrt*(dx\*dx + dy\*dy);

}

// 读入文件，填入List中

**public** **static** List<PP> readPoints(String fname)**throws** Exception{

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(fname)));

List<PP> lst = **new** Vector<PP>();

**for**(;;){

String s = br.readLine();

**if**(s==**null**) **break**;

String[] ss = s.split(",");

PP a = **new** PP(0,0);

a.x = Integer.*parseInt*(ss[0]);

a.y = Integer.*parseInt*(ss[1]);

lst.add(a);

}

br.close();

**return** lst;

}

**public** **static** **void** main(String[] args) **throws** Exception {

List<PP> lst = *readPoints*("in.txt");

**double** x = *f*(lst, **new** RR(0, 0, 1000, 1000));

System.*out*.printf("%.2f",x);

}

}

运行结果：

11.38

44. 递归算法

package temp;

/\*\*

\* 递归算法：将数据分为两部分，递归将数据从左侧移右侧实现全排列

\*

\* @param datas

\* @param target

\*/

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.ArrayList;

**public** **class** T06 {

// 输出

**public** **static** **void** print(List target){

**for**(Object o: target){

System.*out*.print(o);

}

System.*out*.println();

}

// 递归排列

**public** **static** **void** sort(List datas,List target,**int** n){

**if**(target.size()==n){

*print*(target);

**return**;

}

**for**(**int** i=0;i<datas.size();i++){

List newDatas = **new** ArrayList(datas);

List newTarget = **new** ArrayList(target);

newTarget.add(newDatas.get(i));

newDatas.remove(i);

*sort*(newDatas,newTarget,n);

}

}

// 主函数

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

String[] s = {"a","b","c"};

*sort*(Arrays.*asList*(s),**new** ArrayList(),s.length);

}

}

运行结果：

abc

acb

bac

bca

cab

cba

方法二：

**public** **class** AllSort{

**public** **static** **void** perm(String[] buf,**int** start,**int** end){

**if**(start==end){//当只要求对数组中一个字母进行全排列时，只要按该数组输出即可

**for**(**int** i=0;i<=end;i++){

System.*out*.print(buf[i]);

}

System.*out*.println();

}

**else**{//多个字母全排列

**for**(**int** i=start;i<=end;i++){

String temp=buf[start];//交换数组第一个元素与后续的元素

buf[start]=buf[i];

buf[i]=temp;

*perm*(buf,start+1,end);//后续元素递归全排列

temp=buf[start];//将交换后的数组还原

buf[start]=buf[i];

buf[i]=temp;

}

}

}

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

String buf[]={"a","b","c"};

*perm*(buf,0,buf.length-1);

}

}

运行结果：

abc

acb

bac

bca

cba

cab

45.  字符串全拜列

/\*

\* 字符串全拜列

\*/

**public** **class** T03{

// 输出字符数组

**public** **static** **void** print(**char**[] arr){

**for**(**int** i=0;i<arr.length;i++){

System.*out*.print(arr[i]);

}

System.*out*.println();

}

// 递归遍历

**public** **static** **void** perm(**char**[] arr,**int** start,**int** end){

**if**(start==end){

*print*(arr); // 输出

}**else**{

**for**(**int** i=start;i<=end;i++){

// 换位

**char** c = arr[start];

arr[start] = arr[i];

arr[i] = c;

// 递归

*perm*(arr,start+1,end);

// 恢复数组原位

c = arr[start];

arr[start] = arr[i];

arr[i] = c;

}

}

}

// 字符串转字符数组

**public** **static** **void** f(String s){

**char**[] arr = s.toCharArray();

*perm*(arr,0,arr.length-1);

}

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

String s = "abc";

*f*(s);

}

}

运行结果：

abc

acb

bac

bca

cba

cab

46. 亲密数

/\*

假设有a、b两个数，若a的所有因子之和等于b,b的所有因子之和等于a，

并且a不等于b，则称a和b是一对亲密数。如284和220就是一对亲密数。

分析：

若要找出10000以内的亲密数，可使用以下算法：

（1）对每一个数i，将其因子分解出来，并将因子保存到一个数组中，再将因子之和保存到变量b1。

（2）将因子之和b1再进行因子分解，并将因子保存到一个数组中，将因子之和保存到变量b2中。

（3）若b2等于i，并且b1不等于b2，则找到一对亲密数为i和b1，可将其输出。

（4）重复步骤（1）~（3），即可找出指定范围的亲密数。

\*/

**import** java.util.ArrayList;

**import** java.util.Scanner;

**public** **class** T014 {

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

ArrayList a1 = **new** ArrayList();

ArrayList a2 = **new** ArrayList();

**int** b1; // 保存每个数字 i 的所有因子之和

**int** b2; // 保存 b1 的所有因子之和

**int** n; // 查找亲密数的范围 n

Scanner scan = **new** Scanner(System.*in*);

System.*out*.print("输入要查找亲密数的范围：");

n = scan.nextInt();

**for**(**int** i=2; i<n; i++){ // 循环次数

b1 = 0; // 初始化b1 = 0;

**for**(**int** j=1;j<i/2+1;j++){

**if**(i%j==0){

a1.add(j); // 保存 i 的所有因子

b1 += j; // 保存 i 的所有因子之和

}

}

b2 = 0;// 初始化b2 = 0;

// 将 b1 (i 的所有因子之和) 再进行 因子分解

**for**(**int** k=1;k<b1/2+1;k++){

**if**(b1%k==0){

a2.add(k); // 保存 b1 的所有因子

b2 += k; // 保存 b1 的所有因子之和

}

}

//判断 A,B 的输出条件

**if**(b2==i && i<b1){

System.*out*.println(i+"————"+b1 +"是亲密数，各因子为：");

// 输出一个数的所有因子

System.*out*.print(i+"的因子有：");

**for**(**int** x=0;x<a1.size()-1;x++){

System.*out*.print(a1.get(x)+"+");

}

System.*out*.println(a1.get(a1.size()-1)+"="+b1);

// 输出另一个数的所有因子

System.*out*.print(b1+"的因子有：");

**for**(**int** x=0;x<a2.size()-1;x++){

System.*out*.print(a2.get(x)+"+");

}

System.*out*.println(a2.get(a2.size()-1)+"="+b2+"\n");

}

a1.clear(); // 清空数组

a2.clear();

}

}

}

运行结果：

输入要查找亲密数的范围：5000

220————284是亲密数，各因子为：

220的因子有：1+2+4+5+10+11+20+22+44+55+110=284

284的因子有：1+2+4+71+142=220

1184————1210是亲密数，各因子为：

1184的因子有：1+2+4+8+16+32+37+74+148+296+592=1210

1210的因子有：1+2+5+10+11+22+55+110+121+242+605=1184

2620————2924是亲密数，各因子为：

2620的因子有：1+2+4+5+10+20+131+262+524+655+1310=2924

2924的因子有：1+2+4+17+34+43+68+86+172+731+1462=2620