# 打印三角形

如下效果：



public class Demo **{**

static int ROW **=** 30**;**

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

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

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

printChars**(**' '**,**ROW**-**i**);**

printChars**(**'\*'**,**i**\***2**+**1**);**

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

**}**

**}**

static void printChars**(**char ch**,** int n**)** **{**

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

System**.**out**.**print**(**ch**);**

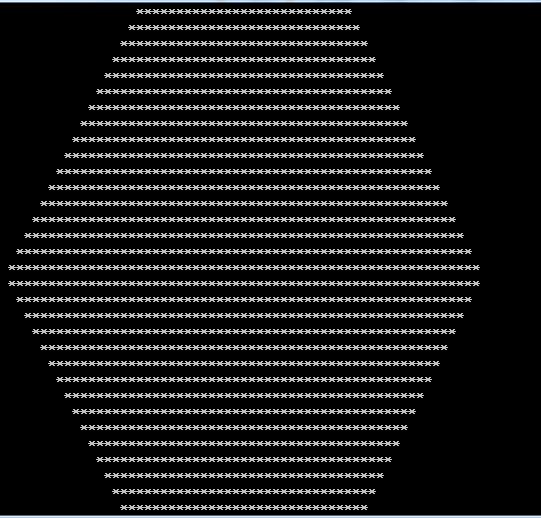
**}**

**}**

**}**

如下效果：

截图只截了中间一部分。



public class Demo **{**

static int ROW **=** 30**;**

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

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

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

printChars**(**' '**,**ROW**-**i**);**

printChars**(**'\*'**,**i**\***2**+**1**);**

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

**}**

**for** **(**int i**=**ROW**-**1**;** i**>=**0**;** i**--)** **{**

printChars**(**' '**,**ROW**-**i**);**

printChars**(**'\*'**,**i**\***2**+**1**);**

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

**}**

**}**

static void printChars**(**char ch**,** int n**)** **{**

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

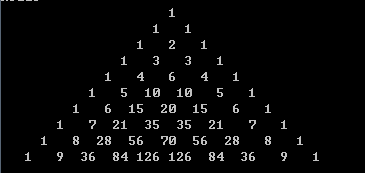
System**.**out**.**print**(**ch**);**

**}**

**}**

**}**

# 杨辉三角，递归



public class Demo **{**

//ROW表示要打印杨辉三角的行数，LEN表示一个数字占用的字符宽度

static int ROW **=** 15**,** LEN **=** 4**;**

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

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

printYangHui**(**ROW**);**

**}**

//打印一个 n 行的杨辉三角

static void printYangHui**(**int n**)** **{**

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

printRow**(**i**);**

**}**

**}**

//打印杨辉三角中第n行的值

static void printRow**(**int n**)** **{**

printSpaces**((**ROW**-**n**)\***LEN**/**2**);**

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

**if** **(**i**==**1 **||** i**==**n**)** **{**

System**.**out**.**printf**(**"%4d"**,** 1**);**/\*4应该和LEN相等，且LEN最好为偶数\*/

**}** **else** **{**

System**.**out**.**printf**(**"%4d"**,** getPoint**(**n**,**i**));**

**}**

**}**

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

**}**

//打印n个空格

static void printSpaces**(**int n**)** **{**

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

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

**}**

**}**

//起始坐标为 1,1

//使用递归求杨辉三角中 某个坐标的值 x行 y列

static int getPoint**(**int x**,** int y**)** **{**

**if** **(**x **==** y **||** y **==** 1**)** **{**

**return** 1**;**

**}** **else** **{**

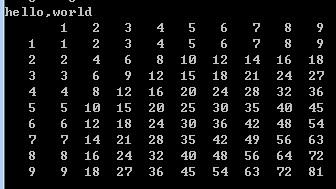
**return** getPoint**(**x**-**1**,**y**-**1**)** **+** getPoint**(**x**-**1**,** y**);**

**}**

**}**

**}**

# 打印9\*9乘法表



public class Demo **{**

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

System**.**out**.**println**(**"hello,world"**);**

print9\_9Table**();**

**}**

static void print9\_9Table**()** **{**

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

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

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

**if** **(**row **==** 0 **&&** column **==** 0**)** **{**

System**.**out**.**printf**(**"%4s"**,** " "**);**

**continue;**//乘法表左上角为空

**}**

**if** **(**row **==** 0 **)** **{**

tb**[**row**][**column**]** **=** column**;**

System**.**out**.**printf**(**"%4d"**,** tb**[**row**][**column**]);**

**continue;**

**}**

**if** **(**column **==** 0**)** **{**

tb**[**row**][**column**]** **=** row**;**

System**.**out**.**printf**(**"%4d"**,** tb**[**row**][**column**]);**

**continue;**

**}**

System**.**out**.**printf**(**"%4d"**,** tb**[**0**][**column**]\***tb**[**row**][**0**]);**

**}**

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

**}**

**}**

**}**

# 10个小朋友围一圈，从1报到99，谁报到几就给谁发几个糖果，求最后谁分的糖果最多？

public class Demo **{**

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

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

getMost**();**

**}**

static void getMost**()** **{**

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

int index**=**0**;**//0~9

**for** **(**int i**=**1**;** i**<=**99**;** i**++)** **{**

childs**[**index**]** **+=** i**;**

index**++;**

**if** **(**index **==** 10**)** index **=** 0**;**

**}**

System**.**out**.**print**(**"ten childs:"**);**

**for** **(**int c **:** childs**)** **{**

System**.**out**.**printf**(**"%4d"**,** c**);**

**}**

int tempValue**=**0**,**ti**=**0**;**

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

**if** **(**tempValue **<** childs**[**i**])** **{**

tempValue **=** childs**[**i**];**

ti **=** i**;**

**}**

**}**

System**.**out**.**printf**(**"\nthe most:child[%d]:%d"**,**ti**,**tempValue**);**

**}**

**}**

# 打印日历

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

public class Demo **{**

static int**[]** monthDays **=** **{**31**,**28**,**31**,**30**,**31**,**30**,**

31**,**31**,**30**,**31**,**30**,**31**};**

static String**[]** monthName **=** **{**"January"**,** "February"**,** "March"**,** "April"**,**

"May"**,** "June"**,** "July"**,** "August"**,**

"September"**,** "October"**,** "November"**,** "December"**};**

static int daysIndex**=**1**;**//1~7

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

System**.**out**.**print**(**"please enter the year,enter 0 exit:"**);**

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

int y **=** in**.**nextInt**();**

**while(**y **!=** 0**)** **{**

printCal**(**y**);**

System**.**out**.**print**(**"please enter the year,enter 0 exit:"**);**

y **=** in**.**nextInt**();**

**}**

**}**

static void printCal**(**int year**)** **{**

//判断是否为闰年

**if** **(**isLeapYear**(**year**))** monthDays**[**1**]** **=** 29**;**

//判断第一天是周几

daysIndex **=** getWeek**(**year**)** **+** 1**;**

**for** **(**int m**=**1**;** m**<=**12**;** m**++)** **{**

System**.**out**.**printf**(**"\n %s %d\n"**,** monthName**[**m**-**1**],**year**);**

System**.**out**.**printf**(**"%4s%4s%4s%4s%4s%4s%4s\n"**,**

"Son"**,**"Mon"**,**"Tue"**,**"Wed"**,**"Thu"**,**"Fri"**,**"Sat"**);**

printMonth**(**m**);**

**}**

**}**

static void printMonth**(**int m**)** **{**

printSpaces**((**daysIndex**-**1**)\***4**);**

**for** **(**int d**=**1**;** d**<=**monthDays**[**m**-**1**];** d**++)** **{**

System**.**out**.**printf**(**"%4d"**,**d**);**

**if** **(**daysIndex **%** 7 **==** 0**)** **{**

daysIndex **=** 1**;**

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

**}** **else** **{**

daysIndex**++;**

**}**

**}**

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

**}**

static void printSpaces**(**int m**)** **{**

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

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

**}**

**}**

//4年一闰 百年不闰 400年再闰 公转周期：365.242天

//365.24219\*4=1460.96876 闰为1461天，多出0.03124

//闰100次 即400年后多出3.124天

//故规定400中的4个整百年有3个不能闰

static boolean isLeapYear**(**int year**)** **{**

**if** **(**year **%** 4 **==** 0**)** **{**

**if** **(**year **%** 100 **==** 0 **&&** year **%** 400 **!=** 0**)** **{**

**return** **false;**

**}**

**return** **true;**

**}** **else** **{**

**return** **false;**

**}**

**}**

//计算一年的第一天是星期几，公元1年的第一天是星期一

//days % 7 = w

//1~6:周一到周六 0:周日

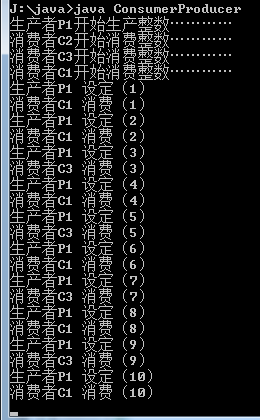
static int getWeek**(**int year**)** **{**

**return** **((**year**-**1**)\***365 **+** **(**year**-**1**)/**4 **+** **(**year**-**1**)/**400 **-** **(**year**-**1**)/**100 **+** 1**)** **%** 7**;**

**}**

**}**

# 消费者生产者问题，线程中synchronized wait notifyAll的使用



**import** java**.**util**.**concurrent**.**TimeUnit**;**

public class ConsumerProducer **{**

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

Clerk clerk **=** **new** Clerk**();**

**new** Thread**(new** Producer**(**clerk**),** "P1"**).**start**();**

**new** Thread**(new** Consumer**(**clerk**),**"C1"**).**start**();**

**new** Thread**(new** Consumer**(**clerk**),**"C2"**).**start**();**

**new** Thread**(new** Consumer**(**clerk**),**"C3"**).**start**();**

**}**

**}**

//店员，每次只能存储一个产品

//生产者向店员提供产品，当店员产品数为1时，等待

//消费者从店员处消费产品，当店员产品为-1（无），等待

class Clerk **{**

private int product **=** **-**1**;**

public synchronized void setProduct**(**int product**)** **throws** InterruptedException **{**

waitFull**();**

**this.**product **=** product**;**

System**.**out**.**printf**(**"生产者%s 设定（%d）\n"**,**Thread**.**currentThread**().**getName**(),**product**);**

notifyAll**();**

**}**

private synchronized void waitFull**()** **throws** InterruptedException **{**

**while** **(this.**product **!=** **-**1**)** **{**

wait**();**

**}**

**}**

public synchronized int getProduct**()** **throws** InterruptedException **{**

TimeUnit**.**SECONDS**.**sleep**(**1**);**

waitIfEmputy**();**

int p **=** **this.**product**;**

**this.**product **=** **-**1**;**

System**.**out**.**printf**(**"消费者%s 消费（%d）\n"**,**Thread**.**currentThread**().**getName**(),** p**);**

notifyAll**();**

**return** p**;**

**}**

private synchronized void waitIfEmputy**()** **throws** InterruptedException **{**

int w **=** 1**;**

**while** **(this.**product **==** **-**1**)** **{**

wait**(**1500**);**

// if (++w > 2) System.exit(0);

**}**

**}**

**}**

class Producer **implements** Runnable **{**

private Clerk clerk**;**

public Producer**(**Clerk clerk**)** **{**

**this.**clerk **=** clerk**;**

**}**

@Override

public void run**()** **{**

System**.**out**.**printf**(**"生产者%s开始生产整数…………\n"**,**Thread**.**currentThread**().**getName**());**

**for** **(**int product**=**1**;** product**<=**10**;** product**++)** **{**

**try** **{**

clerk**.**setProduct**(**product**);**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**}**

**}**

class Consumer **implements** Runnable **{**

private Clerk clerk**;**

public Consumer**(**Clerk clerk**)** **{**

**this.**clerk **=** clerk**;**

**}**

@Override

public void run**()** **{**

System**.**out**.**printf**(**"消费者%s开始消费整数…………\n"**,**Thread**.**currentThread**().**getName**());**

**for** **(**int i**=**1**;** i**<=**10**;** i**++)** **{**

**try** **{**

clerk**.**getProduct**();**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

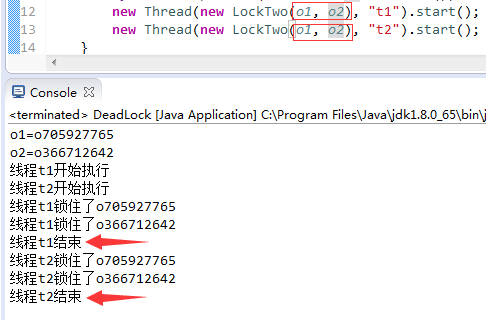
// System.exit(0);

**}**

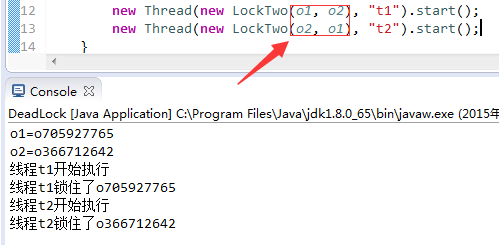
**}**

# 线程死锁现象

没有死锁时：



死锁时：



代码：

**import** java**.**util**.**concurrent**.**TimeUnit**;**

public class DeadLock **{**

static Object o1 **=** **new** Object**();**

static Object o2 **=** **new** Object**();**

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

System**.**out**.**println**(**"o1="**+**"o"**+**o1**.**hashCode**());**

System**.**out**.**println**(**"o2="**+**"o"**+**o2**.**hashCode**());**

**new** Thread**(new** LockTwo**(**o1**,** o2**),** "t1"**).**start**();**

**new** Thread**(new** LockTwo**(**o2**,** o1**),** "t2"**).**start**();**

**}**

**}**

class LockTwo **implements** Runnable **{**

Object o1**,**o2**;**

public LockTwo**(**Object o1**,** Object o2**)** **{**

**this.**o1 **=** o1**;**

**this.**o2 **=** o2**;**

**}**

@Override

public void run**()** **{**

System**.**out**.**println**(**"线程"**+**Thread**.**currentThread**().**getName**()+**"开始执行"**);**

synchronized **(**o1**)** **{**

System**.**out**.**println**(**"线程"**+**Thread**.**currentThread**().**getName**()+**"锁住了o"**+**o1**.**hashCode**());**

work**();**

synchronized **(**o2**)** **{**

System**.**out**.**println**(**"线程"**+**Thread**.**currentThread**().**getName**()+**"锁住了o"**+**o2**.**hashCode**());**

work**();**

**}**

**}**

System**.**out**.**println**(**"线程"**+**Thread**.**currentThread**().**getName**()+**"结束"**);**

**}**

private void work**()** **{**

**try** **{**

Thread**.**sleep**(**3000**);**

**}** **catch** **(**InterruptedException e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**

**}**

**}**

**}**

# 网络下载url.openStream()

**import** java**.**io**.**FileNotFoundException**;**

**import** java**.**io**.**FileOutputStream**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**InputStream**;**

**import** java**.**io**.**OutputStream**;**

**import** java**.**net**.**MalformedURLException**;**

**import** java**.**net**.**URL**;**

public class DownLoad **{**

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

//注意，大部分网站的网页是不能这样直接下载的，需要传入参数

URL url **=** **new** URL**(**"http://openhome.cc/Gossip/Scala/"**);**

**new** DownLoad**().**download**(**url**,** "C:/l.html"**);**

**}**

void download**(**URL url**,** String fileName**)** **{**

**new** Thread**(new** Runnable**()** **{**

public void run**()** **{**

**try** **{**

dump**(**url**.**openStream**(),** **new** FileOutputStream**(**fileName**));**

**}** **catch** **(**FileNotFoundException e**)** **{**

e**.**printStackTrace**();**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**}).**start**();**

**}**

void dump**(**InputStream src **,**OutputStream dest**)** **throws** IOException **{**

**try** **(**InputStream in **=** src**;** OutputStream out **=** dest**)** **{**

byte**[]** buf **=** **new** byte**[**1024**];**

int length**;**

**while** **((**length **=** in**.**read**(**buf**))** **!=** **-**1**)** **{**

out**.**write**(**buf**,** 0**,** length**);**

**}**

**}**

**}**

**}**

【程序1】  
题目：古典问题：有一对兔子，从出生后第3个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，问每个月的兔子总数为多少？  
1.程序分析： 兔子的规律为数列1,1,2,3,5,8,13,21....

# 斐波那契数列

**import** java**.**util**.**Scanner**;**

public class Fibonacci **{**

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

Fibonacci fibonacci **=** **new** Fibonacci**();**

int d**;**

**do** **{**

System**.**out**.**print**(**"enter day, if 0 exit:"**);**

d **=** **new** Scanner**(**System**.**in**).**nextInt**();**

fibonacci**.**printFeibo**(**d**);**

**}** **while** **(**d **!=** 0**);**

**}**

//打印1~n天的兔子数目

void printFeibo**(**int d**)** **{**

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

System**.**out**.**printf**(**"%d "**,** getFibo**(**i**));**

**if** **(**i**%**5**==**0**)**System**.**out**.**println**();**

**}**

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

**}**

//得到第n天的兔子数目

int getFibo**(**int n**)** **{**

**if** **(**n **==** 1 **||** n **==** 2**)** **{**

**return** 1**;**

**}** **else** **{**

**return** getFibo**(**n**-**1**)+**getFibo**(**n**-**2**);**

**}**

**}**

**}**

# 求素数的个数

【程序2】  
题目：判断101-200之间有多少个素数，并输出所有素数。

**素数**，指在大于1的[自然数](https://zh.wikipedia.org/wiki/%E8%87%AA%E7%84%B6%E6%95%B0)中，除了1和该数自身外，无法被其他自然数[整除](https://zh.wikipedia.org/wiki/%E6%95%B4%E9%99%A4)的数，任何大于1的[整数](https://zh.wikipedia.org/wiki/%E6%95%B4%E6%95%B8)均可被表示成一串唯一素数之乘积。验证一个数字 n 是否为素数的一种简单但缓慢的方法为[试除法](https://zh.wikipedia.org/wiki/%E8%A9%A6%E9%99%A4%E6%B3%95)。此一方法会测试 n 是否为任一在2与\sqrt{n}之间的整数之倍数。  
1.程序分析：判断素数的方法：用一个数分别去除2到sqrt(这个数)，如果能被整除，  
      则表明此数不是素数，反之是素数。

public class PrimeNumber **{**

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

int num **=** 0**;**

**for** **(**int i**=**101**;** i**<=**200**;** i**++)** **{**

**if** **(**isPrimeNum**(**i**))** **{**

num**++;**

System**.**out**.**printf**(**"%d "**,**i**);**

**if** **(**num**%**5 **==** 0**)** System**.**out**.**println**();**

**}**

**}**

System**.**out**.**println**(**"\n"**+**"total:"**+**num**);**

**}**

static boolean isPrimeNum**(**int n**)** **{**

**for** **(**int i**=**2**;** i**<=**Math**.**floor**(**Math**.**sqrt**(**n**));** i**++)** **{**

**if** **(**n **%** i **==** 0**)** **return** **false;**

**}**

**return** **true;**

**}**

**}**

# 打印"水仙花数

题目：打印出所有的"水仙花数"，所谓"水仙花数"是指一个三位数，其各位数字立方和等于该数本身。例如：153是一个"水仙花数"，因为153=1的三次方＋5的三次方＋3的三次方。

public class NarcissusFew **{**

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

int num**=**0**;**

**for** **(**int i**=**100**;** i**<=**999**;** i**++)** **{**

**if** **(**isNarcissusFew**(**i**))** **{**

num**++;**

System**.**out**.**printf**(**"%d "**,** i**);**

**if** **(**num **%** 5 **==** 0**)** System**.**out**.**println**();**

**}**

**}**

System**.**out**.**println**(**"\ntotal:"**+**num**);**

**}**

//判断是否为一个水仙花数

static boolean isNarcissusFew**(**int n**)** **{**

**if** **(!(**n **>=** 100 **&&** n **<=** 999**))** **{**

System**.**out**.**println**(**"enter a num between 100~999"**);**

**return** **false;**

**}**

int bai**,**shi**,**ge**;**

bai **=** n**/**100**;**

shi **=** n**%**100**/**10**;**

ge **=** n**%**10**;**

**if** **((**bai**\***bai**\***bai **+** shi**\***shi**\***shi **+** ge**\***ge**\***ge**)** **==** n**)** **{**

**return** **true;**

**}**

**return** **false;**

**}**

**}**

# 整数分解质因数

题目：将一个正整数分解质因数。例如：输入90,打印出90=2\*3\*3\*5。质数（prime number）又称素数。

**import** java**.**awt**.**List**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Scanner**;**

**import** javax**.**naming**.**InsufficientResourcesException**;**

public class PrintPrimeFactor **{**

static java**.**util**.**List**<**Integer**>** ProbablePrimeNum **=** **new** ArrayList**<**Integer**>();**

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

int num**;**

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

System**.**out**.**print**(**"enter a positive num, 0 exit:"**);**

num **=** scanner**.**nextInt**();**

**while** **(**num **!=** 0**)** **{**

printResult2**(**num**);**

System**.**out**.**print**(**"enter a positive num, 0 exit:"**);**

num **=** scanner**.**nextInt**();**

**}**

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

**}**

static void printResult2**(**int n**)** **{**

**if** **(**n **<** 0**)** **{**

System**.**out**.**println**(**"please enter a positive number"**);**

**}** **else** **if** **(**isPrimeNum**(**n**))** **{**

System**.**out**.**println**(**"this is a prime number"**);**

**}** **else** **{**

System**.**out**.**print**(**n**+**"="**);**

int temp **=** n**;**

**for** **(**int i**=**2**;** i**<=** temp**/**2**;** i**++)** **{**

**if** **(**isPrimeNum**(**i**))** **{**

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

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

n **=** n**/**i**;**

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

System**.**out**.**print**(**i**+**"\*"**);**

**}** **else** **{**

System**.**out**.**print**(**i**);**

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

**return;**

**}**

**}** **else** **{**

**break;**

**}**

**}**

**}**

**}**

**}**

**}**

static void printResult**(**int n**)** **{**

**if** **(**isPrimeNum**(**n**))** **{**

System**.**out**.**println**(**"this is a prime number"**);**

**}** **else** **{**

System**.**out**.**print**(**n**+**"="**);**

ProbablePrimeNum **=** getProbablePrimeNum**(**n**);**

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

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

int primeNum **=** ProbablePrimeNum**.**get**(**i**);**

**if** **(**n **%** primeNum **==** 0**)** **{**

n **=** n**/**primeNum**;**

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

System**.**out**.**print**(**primeNum**+**"\*"**);**

**}** **else** **{**

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

**return;**

**}**

**}** **else** **{**

**break;**

**}**

**}**

**}**

**}**

**}**

//判断是否为素数

static boolean isPrimeNum **(**int n**)** **{**

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

int end **=** **(**int**)**Math**.**sqrt**(**n**);**

**for** **(**int i**=**2**;** i**<=**end**;** i**++)** **{**

**if** **(**n **%** i **==** 0**)** **return** **false;**

**}**

**return** **true;**

**}** **else** **{**

**return** **false;**

**}**

**}**

// 得到n的所有可能的质因数

static java**.**util**.**List**<**Integer**>** getProbablePrimeNum**(**int n**)** **{**

java**.**util**.**List**<**Integer**>** list **=** **new** ArrayList**<>();**

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

**if** **(**isPrimeNum**(**i**))** **{**

list**.**add**(**i**);**

**}**

**}**

**return** list**;**

**}**

**}**

# 最大公约数和最小公倍数

题目：输入两个正整数m和n，求其最大公约数和最小公倍数。若n = cd，就说d是n的一个因数（约数），或说n是d的一个倍数

**import** java**.**util**.**Scanner**;**

public class Demo **{**

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

int m**,**n**;**

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

**do** **{**

System**.**out**.**print**(**"enter two number, 0 exit:"**);**

m **=** scanner**.**nextInt**();**

**if** **(**m **==** 0**)** **{**

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

System**.**exit**(**0**);**

**}**

n **=** scanner**.**nextInt**();**

System**.**out**.**println**(**"MaxCommonDivisor:"**+**getMaxCommonDivisor**(**m**,** n**));**

System**.**out**.**println**(**"MinCommonMultiple:"**+**getMinCommonMultiple**(**m**,** n**));**

**}** **while(true);**

**}**

static int getMinCommonMultiple**(**int m**,** int n**)** **{**

**for** **(**int i**=**getMax**(**m**,** n**);** **;** i**++)** **{**

**if** **(**i **%** m **==** 0 **&&** i **%** n **==** 0**)** **{**

**return** i**;**

**}**

**}**

**}**

//最小公倍数

static int getMinCommonMultiple2**(**int m**,** int n**)** **{**

**return** m **\*** n **/** getMaxCommonDivisor**(**m**,** n**);**

**}**

//求最大公因数

static int getMaxCommonDivisor**(**int m**,** int n**)** **{**

int r**=**0**;**

int min **=** getMin**(**m**,** n**);**

int max **=** getMax**(**m**,** n**);**

**if** **(**max **%** min **==** 0**)** **{**

**return** min**;**

**}** **else** **{**

**for** **(**int i**=**1**;** i**<=**min**/**2**;** i**++)** **{**

**if** **(**m **%** i **==** 0 **&&** n **%** i **==** 0**)** **{**

r **=** i**;**

**}**

**}**

**}**

**return** r**;**

**}**

static int getMin**(**int m**,** int n**)** **{**

**return** **(**m **<** n**)** **?** m **:** n**;**

**}**

static int getMax**(**int m**,** int n**)** **{**

**return** **(**m **>** n**)** **?** m **:** n**;**

**}**

**}**

# 字符分类统计

题目：输入一行字符，分别统计出其中 英文字母、空格、数字和其它字符的个数。

**import** java**.**util**.**Scanner**;**

**import** javax**.**swing**.**JOptionPane**;**

public class Count **{**

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

String in**;**

// Scanner scanner = new Scanner(System.in);

// do {

// System.out.print("enter:");

// in = scanner.nextLine();

// statistics(in);

// } while(true);

int option**;**

**do** **{**

in **=** JOptionPane**.**showInputDialog**(null,** "enter:"**,** "enter string dialog"**,** JOptionPane**.**INFORMATION\_MESSAGE**);**

String res **=** statistics**(**in**);**

option **=** JOptionPane**.**showConfirmDialog**(null,** res**,** "do continue?"**,**

JOptionPane**.**YES\_NO\_OPTION**,** JOptionPane**.**INFORMATION\_MESSAGE**);**

**}** **while** **(**option **==** JOptionPane**.**YES\_OPTION**);**

**}**

static String statistics**(**String in**)** **{**

int alpha**=**0**,**digit**=**0**,**space**=**0**,**other**=**0**;**

**if** **(**in **!=** **null)** **{**

char**[]** is **=** in**.**toCharArray**();**

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

char ch **=** is**[**i**];**

**if** **((**ch**>=**'a' **&&** ch**<=**'z'**)** **||** **(**ch**>=**'A'**&&**ch**<=**'Z'**))** **{**

alpha**++;**

**}** **else** **if** **(**ch**>=**'0' **&&** ch**<=**'9'**)** **{**

digit**++;**

**}** **else** **if** **(**ch **==** ' '**)** **{**

space**++;**

**}** **else** **{**

other**++;**

**}**

**}**

**}**

System**.**out**.**println**(**"alpha:"**+**alpha**+**" digit:"**+**digit**+**" space:"

**+**space**+**" other:"**+**other**);**

**return** **(**"alpha:"**+**alpha**+**" digit:"**+**digit**+**" space:"

**+**space**+**" other:"**+**other**);**

**}**

**}**

# s=a+aa+aaa+aaaa+aa...a

题目：求s=a+aa+aaa+aaaa+aa...a的值，其中a是一个数字。例如2+22+222+2222+22222(此时共有5个数相加)，  
几个数相加有键盘控制。

**import** java**.**util**.**Scanner**;**

**import** javax**.**swing**.**JOptionPane**;**

**import** javax**.**swing**.**RowFilter**;**

//求s=a+aa+aaa+aaaa+aa...a的值,n表示相加数值的个数

public class Sum **{**

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

int opition **=** **-**1**;**

**do** **{**

String res **=** JOptionPane**.**showInputDialog**(null,**

"enter a and n,such as:2 3\nsum = 2+22+222,if a=0 exit:"**,**

"sum"**,** JOptionPane**.**INFORMATION\_MESSAGE**);**

**if** **(**res **==** **null)** System**.**exit**(**0**);**

Scanner scanner **=** **new** Scanner**(**res**);**

int a **=** scanner**.**nextInt**();**

**if** **(**a **==** 0**)**System**.**exit**(**0**);**

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

int sum **=** getSum**(**a**,** n**);**

opition **=** JOptionPane**.**showConfirmDialog**(null,** "sum="**+**sum**,** "sum"**,**

JOptionPane**.**YES\_NO\_OPTION**,** JOptionPane**.**INFORMATION\_MESSAGE**);**

**}** **while** **(**opition **==** JOptionPane**.**YES\_OPTION**);**

**}**

static int getSum**(**int a**,** int n**)** **{**

int sum **=** 0**;**

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

sum **+=** getNum**(**a**,** i**);**

**}**

**return** sum**;**

**}**

//获取 aaa…a n为a的个数

static int getNum**(**int a**,** int n**)** **{**

int res**=**0**;**

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

res **=**res **+** a**\***ten**(**i**);**

**}**

**return** res**;**

**}**

//10的n次方

private static int ten**(**int n**)** **{**

int res**=**1**;**

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

res **\*=** 10**;**

**}**

**return** res**;**

**}**

**}**

# 找1000内的完数

题目：一个数如果恰好等于它的因子之和，这个数就称为"完数"。例如6=1＋2＋3.编程 找出1000以内的所有完  
数。

完全数（Perfect number），又称[完美数](http://baike.baidu.com/view/47771.htm)或[完备数](http://baike.baidu.com/view/1275320.htm)，是一些特殊的自然数。如果一个数恰好等于它的因子之和，则称该数为“完全数”[1]  。各个小于它的[约数](http://baike.baidu.com/view/461750.htm)（真约数,列出某数的约数，去掉该数本身，剩下的就是它的真约数）的和等于它本身的自然数叫做**完全数**（Perfect number），又称完美数或完备数。

例如：第一个完全数是6，它有约数1、2、3、6，除去它本身6外，其余3个数相加，1+2+3＝6，恰好等于本身。第二个完全数是28，它有约数1、2、4、7、14、28，除去它本身28外，其余5个数相加，1+2+4+7+14＝28，也恰好等于本身。

public class PerfectNum **{**

static int N **=** 10000**;**

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

System**.**out**.**println**(**getAllPerfectNum**(**N**));**

**}**

static String getAllPerfectNum**(**int n**)** **{**

String res**=**""**;**

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

**if** **(**isPerfectNum**(**i**))** **{**

res **+=** i **+** " "**;**

**}**

**}**

**return** res**;**

**}**

static boolean isPerfectNum**(**int n**)** **{**

int sum**=**0**;**

**for** **(**int i**=**1**;** i**<=**n**/**2**;** i**++)** **{**

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

sum **+=** i**;**

**}**

**}**

**if** **(**sum **==** n**)** **return** **true;**

**return** **false;**

**}**

**}**

# 小球反弹高度

题目：一球从100米高度自由落下，每次落地后反跳回原高度的一半；再落下，求它在 第10次落地时，共经过多  
少米？第10次反弹多高？

**import** javax**.**swing**.**JOptionPane**;**

public class High **{**

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

int opition**;**

**do** **{**

String in **=** JOptionPane**.**showInputDialog**(null,** "输入第几次弹起："**,**

"小球反弹高度计算"**,** JOptionPane**.**INFORMATION\_MESSAGE**);**

int n **=** Integer**.**valueOf**(**in**);**

double high **=** getHigh**(**n**);**

opition **=** JOptionPane**.**showConfirmDialog**(null,**"高度："**+**high**,**"结果"**,**

JOptionPane**.**YES\_NO\_OPTION**,** JOptionPane**.**INFORMATION\_MESSAGE**);**

**}** **while** **(**opition **==** JOptionPane**.**YES\_OPTION**);**

**}**

//n=1 50,n=2,25……

static double getHigh**(**int n**)** **{**

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

**return** 50.0**;**

**}** **else** **{**

**return** getHigh**(**n**-**1**)** **/** 2**;**

**}**

**}**

**}**

# 4个数字组成不重复的三位数

题目：有1、2、3、4个数字，能组成多少个互不相同且无重复数字的三位数？都是多少？

**import** javax**.**swing**.**JOptionPane**;**

public class GetNum **{**

static int**[]** digit **=** **new** int**[]** **{**1**,**2**,**3**,**4**};**

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

JOptionPane**.**showMessageDialog**(null,** getNums**(),** "1,2,3,4组成不重复三位数"

**,** JOptionPane**.**INFORMATION\_MESSAGE**);**

**}**

static String getNums**()** **{**

StringBuilder res **=** **new** StringBuilder**();**

int bai**,**shi**,**ge**,**n**=**0**,**num**;**

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

bai **=** digit**[**i**];**

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

shi **=** digit**[**j**];**

**if** **(**shi **!=** bai**)** **{**

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

ge **=** digit**[**k**];**

**if** **(**ge **!=** shi **&&** ge **!=** bai**)** **{**

n**++;**

num **=** bai**\***100 **+** shi**\***10 **+** ge**;**

res**.**append**(**num**).**append**(**" "**);**

**if** **(**n **%** 5 **==** 0**)**res**.**append**(**"\n"**);**

**}**

**}**

**}**

**}**

**}**

res**.**insert**(**0**,**"total:"**+**n**+**"\n"**);**

**return** res**.**toString**();**

**}**

**}**

# 找数字n，n+100,n+168是平方数

题目：一个整数，它加上100后是一个完全平方数，再加上168又是一个完全平方数，请问该数是多少？

**import** javax**.**swing**.**JOptionPane**;**

public class GetNum **{**

static int**[]** digit **=** **new** int**[]** **{**1**,**2**,**3**,**4**};**

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

JOptionPane**.**showMessageDialog**(null,** getNums**(),** ""

**,** JOptionPane**.**INFORMATION\_MESSAGE**);**

**}**

static String getNums**()** **{**

int i**=**0**;**

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

i**++;**

**if** **(**isSqure**(**i**+**100**)** **&&** isSqure**(**i**+**168**))** **{**

**break;**

**}**

**}**

**return** i**+**""**;**

**}**

//判断一个数是不是整数平方数

static boolean isSqure**(**int n**)** **{**

int squr **=** **(**int**)** Math**.**sqrt**(**n**);**

**if** **(**n **==** squr**\***squr**)** **return** **true;**

**return** **false;**

**}**

**}**

# 判断某天是这一年的第几天

题目：输入某年某月某日，判断这一天是这一年的第几天？

**import** java**.**util**.**Scanner**;**

**import** javax**.**swing**.**JOptionPane**;**

public class CalDay **{**

static int**[]** months **=** **new** int**[]** **{**31**,**28**,**31**,**30**,**31**,**30**,**

31**,**31**,**30**,**31**,**30**,**31**};**

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

String in **=** JOptionPane**.**showInputDialog**(null,** "输入，如：2011 4 3"**);**

JOptionPane**.**showMessageDialog**(null,** getDay**(**in**),** "计算天数"

**,** JOptionPane**.**INFORMATION\_MESSAGE**);**

**}**

static int getDay**(**String in**)** **{**

Scanner scanner **=** **new** Scanner**(**in**);**

int y **=** scanner**.**nextInt**();**

int m **=** scanner**.**nextInt**();**

int d **=** scanner**.**nextInt**();**

int days**=**0**;**

**for** **(**int i**=**0**;** i**<**m**-**1**;** i**++)** **{**

days **+=** months**[**i**];**

**}**

days **+=** d**;**

**if** **(**isLeapYear**(**y**)** **&&** m **>** 2**)** days**++;**

**return** days**;**

**}**

//判断闰年

static boolean isLeapYear**(**int year**)** **{**

**if** **((**year **%** 4 **==** 0 **&&** year **%** 100 **!=** 0**)** **||** year **%** 400 **==** 0**)**

**return** **true;**

**return** **false;**

**}**

**}**

# 三个数由小到大输出

题目：输入三个整数x,y,z，请把这三个数由小到大输出。

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

**import** javax**.**swing**.**JOptionPane**;**

public class PrintNums **{**

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

String in **=** JOptionPane**.**showInputDialog**(null,** "输入3个数字"**);**

String res **=** sort2**(**in**);**

JOptionPane**.**showMessageDialog**(null,** res**);**

**}**

//对输入的3个数字排序

static String sort**(**String in**)** **{**

String res **=** ""**;**

Scanner scanner **=** **new** Scanner**(**in**);**

int a **=** scanner**.**nextInt**();**

int b **=** scanner**.**nextInt**();**

int c **=** scanner**.**nextInt**();**

**if** **(**a **>** b**)** **{**

int temp **=** a**;**

a **=** b**;**

b **=** temp**;**

**}**

**if** **(**c **<** a**)** **{**

res **=** c **+**" "**+**a**+**" "**+**b**;**

**}** **else** **{**

**if** **(**c **>** b**)** **{**

res **=** a**+** " "**+**" "**+**b**+**" "**+**c**;**

**}** **else** **{**

res **=** a**+**" "**+**c**+**" "**+**b**;**

**}**

**}**

**return** res**;**

**}**

//对输入的任意个数字排序

static String sort2**(**String in**)** **{**

String res **=** ""**;**

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

Scanner scanner **=** **new** Scanner**(**in**);**

**while** **(**scanner**.**hasNextInt**())** **{**

nums**.**add**(**scanner**.**nextInt**());**

**}**

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

int k **=** i**;**

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

**if** **(**nums**.**get**(**k**)** **>** nums**.**get**(**j**))** **{**

k **=** j**;**

**}**

**}**

**if** **(**k **!=** i**)** **{**

int temp **=** nums**.**get**(**k**);**

nums**.**set**(**k**,** nums**.**get**(**i**));**

nums**.**set**(**i**,** temp**);**

**}**

**}**

**for** **(**Integer i **:** nums**)** **{**

System**.**out**.**print**(**i**+**" "**);**

res **+=** i**+**" "**;**

**}**

**return** res**;**

**}**

**}**

# 猴子吃桃问题

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

**import** javax**.**swing**.**JOptionPane**;**

public class Monkey **{**

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

String in **=** JOptionPane**.**showInputDialog**(null,** "输入第几天"**);**

JOptionPane**.**showMessageDialog**(null,**

"这一天的桃子个数:"**+**getPeach2**(**Integer**.**valueOf**(**in**)));**

**}**

static int getPeach**(**int day**)** **{**

**if** **(**day **>** 10 **||** day **<** 1**)** **return** 0**;**

**if** **(**day **==** 10**)** **{**

**return** 1**;**

**}** **else** **{**

**return** **(**getPeach**(**day**+**1**)+**1**)\***2**;**

**}**

**}**

static int getPeach2**(**int day**)** **{**

int peachs **=** 1**;**

**if** **(**day **>** 10 **||** day **<** 1**)** **return** 0**;**

**for** **(**int i**=**9**;** i**>=**day**;** i**--)** **{**

peachs **=** **(**peachs**+**1**)\***2**;**

**}**

**return** peachs**;**

**}**

**}**

# 乒乓球队进行比赛

题目：两个乒乓球队进行比赛，各出三人。甲队为a,b,c三人，乙队为x,y,z三人。已抽签决定比赛名单。有人向队员打听比赛的名单。a说他不和x比，c说他不和x,z比，请编程序找出三队赛手的名单。

**import** javax**.**swing**.**JOptionPane**;**

public class Game **{**

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

JOptionPane**.**showMessageDialog**(null,** printMatch**());**

**}**

static String printMatch**()** **{**

StringBuilder all **=** **new** StringBuilder**();**

String right **=** **null;**

//xyz数组代表 x,y,z三个队员，初始值表示未分配对手，

//1值表示已经分配对手

char**[]** xyz **=** **new** char**[]** **{**'x'**,**'y'**,**'z'**};**

char a**,**b**,**c**;**

**for** **(**int i**=**0**;** i**<**3**;** i**++)** **{**

a **=** xyz**[**i**];**//为a分配对手

**for** **(**int j**=**0**;** j**<**3**;** j**++)** **{**

//为b分配对手

b **=** xyz**[**j**];**

**if** **(**b **!=** a**)** **{**

**for** **(**int k**=**0**;** k**<**3**;** k**++)** **{**

c **=** xyz**[**k**];**//为c分配对手

**if** **(**c **!=** b **&&** c **!=** a**)** **{**//可行分组

all**.**append**(**String**.**format**(**"a--%c,b--%c,c--%c\n"

**,**a**,**b**,**c**));**

//条件限制

**if** **(**a **!=** 'x' **&&** c **!=** 'x' **&&** c **!=** 'z'**)**

right **=** String**.**format**(**"a--%c,b--%c,c--%c\n"

**,**a**,**b**,**c**);**

**break;**

**}**

**}**

**}**

**}**

**}**

all**.**insert**(**0**,** "所有分组：\n"**);**

all**.**append**(**"要求的分组：\n"**+**right**);**

**return** all**.**toString**();**

**}**

**}**

# 数列的前20项之和

题目：有一分数序列：2/1，3/2，5/3，8/5，13/8，21/13...求出这个数列的前20项之和。

**import** javax**.**swing**.**JOptionPane**;**

public class CalSum **{**

//分数序列：2/1，3/2，5/3，8/5，13/8，21/13...

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

JOptionPane**.**showMessageDialog**(null,** calSum**(**20**));**

**}**

static String calSum**(**int n**)** **{**

double sum **=** 0**;**

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

sum **+=** getNum**(**i**);**

**}**

**return** String**.**format**(**"%.5f"**,** sum**);**

**}**

//获取分数序列中的单个分数项

static double getNum**(**int n**)** **{**

**return** getSequence**(**n**+**1**)/(**double**)**getSequence**(**n**);**

**}**

//获取 1 2 3 5 8……数列

static int getSequence**(**int n**)** **{**

**if** **(**n **==** 1**)** **return** 1**;**

**if** **(**n **==** 2**)** **return** 2**;**

**return** getSequence**(**n**-**1**)+**getSequence**(**n**-**2**);**

**}**

**}**

# 求1+2!+3!+...+20!的和

题目：求1+2!+3!+...+20!的和

**import** javax**.**swing**.**JOptionPane**;**

public class CalSum **{**

//求1+2!+3!+...+20!的和

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

// System.out.println(getNum(20));

JOptionPane**.**showMessageDialog**(null,** calSum**(**20**));**

**}**

static String calSum**(**int n**)** **{**

long sum **=** 0**;**

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

sum **+=** getNum**(**i**);**

**}**

**return** String**.**format**(**"%d"**,** sum**);**

**}**

//求n的阶乘

static long getNum**(**int n**)** **{**

**if** **(**n **==** 1**)** **return** 1**;**

**return** **(**long**)**n**\***getNum**(**n**-**1**);**

**}**

**}**

# 递归求年龄

题目：有5个人坐在一起，问第五个人多少岁？他说比第4个人大2岁。问第4个人岁数，他说比第3个人大2岁。问第三个人，又说比第2人大两岁。问第2个人，说比第一个人大两岁。最后问第一个人，他说是10岁。请问第五个人多大？

**import** javax**.**swing**.**JOptionPane**;**

public class GetAge **{**

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

JOptionPane**.**showMessageDialog**(null,** getAge**(**2**));**

**}**

static int getAge**(**int n**)** **{**

**if** **(**n **==** 1**)** **return** 10**;**

**return** 2 **+** getAge**(**n**-**1**);**

**}**

**}**

# 正整数分解

题目：给一个不多于5位的正整数，要求：一、求它是几位数，二、逆序打印出各位数字。

扩展为可输入任意小于int最大值的正整数

**import** javax**.**swing**.**JOptionPane**;**

public class ResolveDigit **{**

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

String in **=** JOptionPane**.**showInputDialog**(**"输入任意数字"**);**

JOptionPane**.**showMessageDialog**(null,** "原来的:"**+**in

**+**"\n转换后:"**+**resolve**(**Integer**.**valueOf**(**in**)));**

**}**

//规律:4=4321%10000/1000;3=4321%1000/100;2=4321%100/10;1=4321%10/1;

static String resolve**(**int num**)** **{**

StringBuilder res **=** **new** StringBuilder**();**

int**[]** digit **=** **new** int**[**20**];**//存放输入数字的每一位

int n **=** judgeDigit**(**num**);**

**for** **(**int i**=**n**;** i**>=**1**;** i**--)** **{**

digit**[**i**-**1**]** **=** num **%** powerOfTen**(**i**)/**powerOfTen**(**i**-**1**);**

**}**

//分解出来的整数 每一位逆序存放在数组

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

res**.**append**(**digit**[**i**]);**

**}**

**return** res**.**toString**();**

**}**

//判断数字是几位数字

static int judgeDigit**(**int num**)** **{**

int n **=** 1**;**

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

**if** **(**powerOfTen**(**n**)** **>** num**)** **{**

**break;**

**}** **else** **{**

n**++;**

**}**

**}**

**return** n**;**

**}**

//获取10的n次方

static int powerOfTen**(**int n**)** **{**

**if** **(**n **==** 0**)** **return** 1**;**

**return** 10**\***powerOfTen**(**n**-**1**);**

**}**

**}**

# 判断回文

题目：一个5位数，判断它是不是回文数。即12321是回文数，个位与万位相同，十位与千位相同。

升级为能对任意长度的数字，或者字符串判断。

**import** javax**.**swing**.**JOptionPane**;**

public class JudgeHeadTail **{**

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

String in **=** JOptionPane**.**showInputDialog**(**"输入任意字符串或者数字"**);**

JOptionPane**.**showMessageDialog**(null,** "是否为回文："**+**isSame**(**in**));**

**}**

//判断是否为回文

static boolean isSame**(**String instr**)** **{**

char **[]** in **=** getChars**(**instr**);**

int len **=** in**.**length**;**

int i**;**

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

//首尾判断

**if** **(**in**[**i**]** **!=** in**[**len**-**1**-**i**])** **{**

**break;**

**}**

**}**

**if** **(**i **==** len**/**2**)** **{**

**return** **true;**

**}**

**return** **false;**

**}**

//转换为字符数组

static char**[]** getChars**(**String in**)** **{**

**return** in**.**toCharArray**();**

**}**

**}**

# 判断星期几

题目：请输入星期几的第一个字母来判断一下是星期几，如果第一个字母一样，则继续   判断第二个字母。

package com**.**zht**;**

**import** java**.**awt**.**BorderLayout**;**

**import** javax**.**swing**.**JFrame**;**

**import** javax**.**swing**.**JLabel**;**

**import** javax**.**swing**.**JPanel**;**

**import** javax**.**swing**.**JTextArea**;**

**import** javax**.**swing**.**JTextField**;**

**import** javax**.**swing**.**event**.**CaretEvent**;**

**import** javax**.**swing**.**event**.**CaretListener**;**

public class JudgeWeekend **extends** JFrame**{**

static String**[]** week **=** **{**"Sunday(日)"**,**"Monday(一)"**,**"Tuesday(二)"

**,**"Wednesday(三)"**,**"Thursday(四)"**,**"Friday(五)"**,**"Saturday(六)"**};**

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

**new** JudgeWeekend**();**

System**.**out**.**println**(**whatDay2**(**"t"**));**

**}**

public JudgeWeekend**()** **{**

setLayout**(new** BorderLayout**());**

JLabel label1 **=** **new** JLabel**(**"输入星期："**);**

JTextField input **=** **new** JTextField**(**20**);**

JPanel inPanel **=** **new** JPanel**(new** BorderLayout**());**

inPanel**.**add**(**label1**,** BorderLayout**.**NORTH**);**

inPanel**.**add**(**input**,** BorderLayout**.**CENTER**);**

JLabel label2 **=** **new** JLabel**(**"查询结果："**);**

JTextArea res **=** **new** JTextArea**(**3**,** 20**);**

res**.**setEditable**(false);**

JPanel outPanel **=** **new** JPanel**(new** BorderLayout**());**

outPanel**.**add**(**label2**,** BorderLayout**.**NORTH**);**

outPanel**.**add**(**res**,** BorderLayout**.**CENTER**);**

input**.**addCaretListener**(new** CaretListener**()** **{**

@Override

public void caretUpdate**(**CaretEvent e**)** **{**

JTextField textField **=** **(**JTextField**)** e**.**getSource**();**

String in **=** textField**.**getText**();**

//输入超过2个字符就不在判断

**if** **(**in **!=** **null** **&&** in**.**length**()** **>** 0**)** **{**

res**.**setText**(**whatDay2**(**in**));**

**}**

**}**

**});**

add**(**inPanel**,** BorderLayout**.**NORTH**);**

add**(**outPanel**,** BorderLayout**.**CENTER**);**

pack**();**

setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

setLocationRelativeTo**(null);**

setVisible**(true);**

**}**

//根据输入字符串判断

static String whatDay**(**String in**)** **{**

StringBuilder res **=** **new** StringBuilder**();**

String inLow **=** in**.**toLowerCase**();**

**for** **(**int i**=**0**;** i**<**7**;** i**++)** **{**

//使用每个字符串的前2个子串判断，但有个限制，对输入字符的长度要小于2

**if** **(**week**[**i**].**toLowerCase**().**substring**(**0**,** 2**).**contains**(**inLow**))** **{**

res**.**append**(**week**[**i**]+**"\n"**);**

**}**

**}**

**return** res**.**toString**();**

**}**

//方法2

static String whatDay2**(**String in**)** **{**

StringBuilder res **=** **new** StringBuilder**();**

int len **=** in**.**length**();**

String inLow **=** in**.**toLowerCase**();**

**for** **(**int i**=**0**;** i**<**7**;** i**++)** **{**

int index **=** 0**;**

**while** **(**index **<** len**)** **{**

**if** **(**week**[**i**].**toLowerCase**().**charAt**(**index**)**

**!=** inLow**.**charAt**(**index**))** **{**

**break;**

**}** **else** **{**

index**++;**

**}**

**}**

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

**if** **(**index **==** len**)** **{**

res**.**append**(**week**[**i**]** **+** "\n"**);**

**}**

**}**

**return** res**.**toString**();**

**}**

//根据第一个字母判断星期几

static String whatDay**(**char ch**)** **{**

StringBuilder res **=** **new** StringBuilder**();**

//转换为小写

char chLow **=** charToLowCase**(**ch**);**

**for** **(**int i**=**0**;** i**<**7**;** i**++)** **{**

**if** **(**chLow **==** charToLowCase**(**week**[**i**].**charAt**(**0**)))** **{**

res**.**append**(**week**[**i**]+**"\n"**);**

**}**

**}**

**return** res**.**toString**();**

**}**

//将字符转换为小写

static char charToLowCase**(**char ch**)** **{**

**return** **((**ch**+**""**).**toLowerCase**()).**charAt**(**0**);**

**}**

**}**

# 汉诺塔问题

有三根杆子A，B，C。A杆上有N个(N>1)穿孔圆盘，盘的尺寸由下到上依次变小。要求按下列规则将所有圆盘移至C杆：

1. 每次只能移动一个圆盘；
2. 大盘不能叠在小盘上面。

提示：可将圆盘临时置于B杆，也可将从A杆移出的圆盘重新移回A杆，但都必须遵循上述两条规则。

问：如何移？最少要移动多少次？

public class HanNuo **{**

static int steps **=** 0**;**

static int N **=** 5**;**

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

move**(**N**,** "A"**,** "B"**,** "C"**);**

**}**

//将n个盘子从 src借助between移动到 dest

static void move**(**int n**,** String src**,**String between**,**String dest**)** **{**

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

singleStep**(**n**,** src**,** dest**);**

**}** **else** **{**

move**(**n**-**1**,** src**,** dest**,** between**);**

singleStep**(**n**,** src**,** dest**);**

move**(**n**-**1**,** between**,** src**,** dest**);**

**}**

**}**

//将第n个盘子从 src 移动到 dest

static void singleStep**(**int n**,** String src**,** String dest**)** **{**

steps**++;**

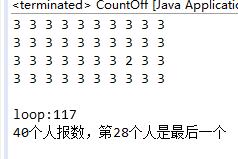
System**.**out**.**printf**(**"第%d步，第%d个盘子：%s-->%s\n"**,**steps**,**n**,**src**,**dest**);**

**}**

**}**

# n个人一圈报数问题

题目：有n个人围成一圈，顺序排号。从第一个人开始报数（从1到3报数），凡报到3的人退出圈子，问最后留下的是原来第几号的那位。



package com**.**zht**;**

/\*\*

\* @author Administrator

\*N个人由N个Int型数值表示，每个Int值代表一个人

\*每个人报的数存放在Int值中，可能为1~COUNT

\*如果代表某人的Int值为COUNT，则表示退出圈子，不再参与报数

\*一直循环，对所有非COUNT值的人报数，直到剩下最后一个人为止

\*/

public class CountOff **{**

//N表示报数的人数

//COUNT表示从1报到COUNT

//loop表示报到剩最后一个人时，总共报了多少次数

static int N **=** 40 **,** COUNT **=** 3**,** loop**=**0**;**

static int**[]** people **=** **new** int**[**N**];**

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

int i **=** countOff**(**people**);**

print**(**people**);**

System**.**out**.**println**(**"loop:"**+**loop**);**

System**.**out**.**printf**(**"%d个人报数，第%d个人是最后一个"**,**N**,**i**+**1**);**

**}**

static int countOff**(**int**[]** people **)** **{**

int len **=** people**.**length**;**

int off **=** 0**;**//退出圈子的人数

int c **=** 1**;**//报数值 1~COUNT

int i **=** 0**;**

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

**if** **(**people**[**i**]** **!=** COUNT**)** **{**

people**[**i**]** **=** c**++;**

loop**++;**

**if** **(**c **==** COUNT**+**1**)** **{**

c **=** 1**;**

off**++;**//报到最后一个数COUNT，表明有一人退出

**if** **(**off **==** len**-**1**)** **break;**

**}**

**}**

i**++;**

**if** **(**i **==** len**)** i**=**0**;**

**}**

i **=** 0**;**

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

**if** **(**people**[**i**]** **!=** COUNT**)** **break;**

i**++;**

**}**

**return** i**;**

**}**

static void print**(**int**[]** people**)** **{**

int n **=** 1**;**

**for** **(**int i **:** people**)** **{**

System**.**out**.**printf**(**"%d "**,**i**);**

**if** **(**n**++** **%** 10 **==** 0**)**

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

**}**

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

**}**

**}**

# 偶数总能表示为两个素数之和

题目：一个偶数总能表示为两个素数之和。

public class ResolveToPrimeNum **{**

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

resolve**(**5000**);**

**}**

static void resolve**(**int n**)** **{**

**if** **(**n **%** 2 **==** 0**)** **{**

int one**,**two**;**

**for** **(**one**=**2**;** one**<**n**;** one**++)** **{**

**if** **(**isPrimeNum**(**one**))** **{**

two **=** n **-** one**;**

**if** **(**isPrimeNum**(**two**))** **{**

System**.**out**.**printf**(**"偶数%d = %d + %d\n"**,**n**,**one**,**two**);**

**break;**

**}**

**}**

**}**

**}** **else** **{**

**return;**

**}**

**}**

static boolean isPrimeNum**(**int n**)** **{**

**for** **(**int i**=**2**;** i**<=**Math**.**floor**(**Math**.**sqrt**(**n**));** i**++)** **{**

**if** **(**n **%** i **==** 0**)** **return** **false;**

**}**

**return** **true;**

**}**

**}**

# NIO和列出文件目录

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.net.URL;

**import** java.nio.channels.Channels;

**import** java.nio.channels.ReadableByteChannel;

**import** java.nio.channels.WritableByteChannel;

**import** java.nio.file.DirectoryStream;

**import** java.nio.file.FileVisitResult;

**import** java.nio.file.Files;

**import** java.nio.file.Path;

**import** java.nio.file.Paths;

**import** java.nio.file.SimpleFileVisitor;

**import** java.nio.file.attribute.BasicFileAttributes;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Date;

**import** java.nio.ByteBuffer;

public class Demo {

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

System.out.println("hello");

listFiles("G:\\Android");

dirAll("H:\\2模拟CMOS");

downLoad("http://openhome.cc", "C:\\cc.html");

}

//列出目录下的文件夹和文件 只查找一级

static void listFiles(String pathStr) **throws** Exception {

java.util.List<String> files = **new** ArrayList<>();

**try** (DirectoryStream<Path> directoryStream =

Files.newDirectoryStream(Paths.get("G:\\Android"))) {

System.out.println("文件夹：");

**for** (Path path : directoryStream) {

**if** (Files.isDirectory(path)) {

System.out.println(path.getFileName());

BasicFileAttributes attributes

= Files.readAttributes(path, BasicFileAttributes.class);

System.out.println("size:"+attributes.size()/1024+"KB");

System.out.println("create time:"+

**new** SimpleDateFormat("yyyy-MM-dd hh:mm:ss").

format(**new** Date(attributes.creationTime().toMillis()))

+"\n");

}

**else** {

files.add(path.getFileName().toString());

BasicFileAttributes attributes

= Files.readAttributes(path, BasicFileAttributes.class);

files.add("size:"+attributes.size()/1024+"KB");

files.add("create time:"+

**new** SimpleDateFormat("yyyy-MM-dd hh:mm:ss").

format(**new** Date(attributes.creationTime().toMillis()))

+"\n");

}

}

System.out.println("\n文件:");

files.forEach(System.out::println);

}

}

//列出目录下所有文件夹和文件

static void dirAll(String ps) {

**try** {

Files.walkFileTree(Paths.get(ps), **new** ConsoleFileVisitor());

} **catch** (IOException e) {

e.printStackTrace();

}

}

//使用NIO

static void downLoad(String src, String dest) {

URL url;

**try** {

url = **new** URL(src);

ReadableByteChannel rByteChannel = Channels.newChannel(url.openStream());

WritableByteChannel wByteChannel = **new** FileOutputStream(dest).getChannel();

ByteBuffer buffer = ByteBuffer.allocate(1024);

**while** (rByteChannel.read(buffer) != -1) {

buffer.flip();

wByteChannel.write(buffer);

buffer.clear();

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

class ConsoleFileVisitor **extends** SimpleFileVisitor<Path> {

@Override

public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attrs)

**throws** IOException {

printSpace(dir);

System.out.printf("[%s]\n", dir.toString());

**return** FileVisitResult.CONTINUE;

}

@Override

public FileVisitResult visitFile(Path file, BasicFileAttributes attrs)

**throws** IOException {

printSpace(file);

System.out.printf("%s%n", file.getFileName());

**return** FileVisitResult.CONTINUE;

}

@Override

public FileVisitResult visitFileFailed(Path file, IOException exc)

**throws** IOException {

System.err.println(exc);

**return** FileVisitResult.CONTINUE;

}

void printSpace(Path path) {

System.out.printf("%"+path.getNameCount()\*2+"s"," ");

}

}

# 小猫钓鱼游戏

游戏规则：一副扑克牌，去掉大小王后平均分配给2个人，轮流出牌成一列，当出现长队中有跟你出的牌相同的牌，这两张牌之间的牌（含这两张牌）你就赢走，最后谁手上没牌为负。

注：有时候会出现无法结束的情况，可根据ws的值进行判断，当大于某个值的时候强行退出游戏。

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

Queue tom **=** **new** Queue**();**

Queue jack **=** **new** Queue**();**

Stack cards **=** **new** Stack**();**

Scanner scanner **=** **new** Scanner**(**"2 4 1 2 5 6"**);**

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

tom**.**add**(**scanner**.**nextInt**());**

**}**

scanner **=** **new** Scanner**(**"3 1 3 5 6 4"**);**

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

jack**.**add**(**scanner**.**nextInt**());**

**}**

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

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

int outCard**,**ws**=**0**;**

**while** **(**tom**.**hasCard**()** **&&** jack**.**hasCard**())** **{**

// jack.playCard(cards);

// if (!tom.hasCard()) break;

// tom.playCard(cards);

outCard **=** tom**.**out**();**

**if** **(!**cards**.**contain**(**outCard**))** **{**

cards**.**push**(**outCard**);**

System**.**out**.**println**(**"\ntom out:"**+**outCard**);**

showInfo**(**tom**,** jack**,** cards**);**

**}** **else** **{**

System**.**out**.**println**(**"\ntom out:"**+**outCard**);**

showInfo**(**tom**,** jack**,** cards**);**

tom**.**add**(**outCard**);**

int winCard **=** cards**.**pop**();**

**while** **(**winCard **!=** outCard**)** **{**

tom**.**add**(**winCard**);**

winCard **=** cards**.**pop**();**

**}**

tom**.**add**(**winCard**);**

ws**++;**

showInfo**(**tom**,** jack**,** cards**);**

**}**

**if** **(!**tom**.**hasCard**())** **break;**

outCard **=** jack**.**out**();**

**if** **(!**cards**.**contain**(**outCard**))** **{**

cards**.**push**(**outCard**);**

System**.**out**.**println**(**"\njack out:"**+**outCard**);**

showInfo**(**tom**,** jack**,** cards**);**

**}** **else** **{**

System**.**out**.**println**(**"\njack out:"**+**outCard**);**

showInfo**(**tom**,** jack**,** cards**);**

jack**.**add**(**outCard**);**

int winCard **=** cards**.**pop**();**

**while** **(**winCard **!=** outCard**)** **{**

jack**.**add**(**winCard**);**

winCard **=** cards**.**pop**();**

**}**

jack**.**add**(**winCard**);**

ws**++;**

showInfo**(**tom**,** jack**,** cards**);**

**}**

**}**

showInfo**(**tom**,** jack**,** cards**);**

**}**

private static void showInfo**(**Queue tom**,** Queue jack**,** Stack cards**)** **{**

System**.**out**.**println**(**"tom:"**+**tom**);**

System**.**out**.**println**(**"jack:"**+**jack**);**

System**.**out**.**println**(**"cards:"**+**cards**);**

**}**

**}**

class Queue **{**

public int**[]** data **=** **new** int**[**1000**];**

public int head **=** 1**;**

public int tail **=** 1**;**

public void add**(**int n**)** **{**

data**[**tail**]** **=** n**;**

tail**++;**

**}**

public int out**()** **{**

int n**;**

n **=** data**[**head**];**

head**++;**

**return** n**;**

**}**

public boolean hasCard**()** **{**

**if** **(**head **>=** tail**)** **return** **false;**

**return** **true;**

**}**

public void playCard**(**Stack cards**)** **{**

int outCard **=** out**();**

**if** **(!**cards**.**contain**(**outCard**))** **{**

cards**.**push**(**outCard**);**

**}** **else** **{**

add**(**outCard**);**

int winCard **=** cards**.**pop**();**

**while** **(**winCard **!=** outCard**)** **{**

add**(**winCard**);**

winCard **=** cards**.**pop**();**

**}**

add**(**winCard**);**

**}**

**}**

@Override

public String toString**()** **{**

StringBuilder sb **=** **new** StringBuilder**();**

// sb.append("head:"+head+" tail:"+tail+"--");

**for** **(**int i**=**head**;** i**<**tail**;** i**++)** **{**

sb**.**append**(**data**[**i**]+**" "**);**

**}**

**return** sb**.**toString**().**trim**();**

**}**

**}**

class Stack **{**

public int**[]** book **=** **new** int**[**1000**];**

public int top **=** 0**;**

public void push**(**int n**)** **{**

top**++;**

book**[**top**]** **=** n**;**

**}**

public int pop**()** **{**

int c **=** book**[**top**];**

top**--;**

**return** c**;**

**}**

public boolean contain**(**int n**)** **{**

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

**if** **(**n **==** book**[**i**])** **return** **true;**

**}**

**return** **false;**

**}**

@Override

public String toString**()** **{**

StringBuilder sb **=** **new** StringBuilder**();**

// sb.append("top:"+top+"--");

**for** **(**int i**=**top**;** i**>**0**;** i**--)** **{**

sb**.**append**(**book**[**i**]+**" "**);**

**}**

**return** sb**.**toString**().**trim**();**

**}**

**}**

# 字符串加密



**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while(**in**.**hasNextLine**()){**

String key **=** in**.**nextLine**();**

String data **=** in**.**nextLine**();**

List**<**Character**>** encrypt **=** **new** ArrayList**<>();**

String keyUp **=** key**.**toUpperCase**();**

String dataUp **=** data**.**toUpperCase**();**

//生成大写字母A~Z 对应的加密替换字母

int offset **=** 0**;**

**for** **(**int i**=**0**;** i**<**26**+**keyUp**.**length**();** i**++)** **{**

**if** **(**i **<** keyUp**.**length**())** **{**

**if** **(!**encrypt**.**contains**(**keyUp**.**charAt**(**i**)))** **{**

encrypt**.**add**(**keyUp**.**charAt**(**i**));**

**}**

**}** **else** **{**

**if** **(!**encrypt**.**contains**((**char**)(**'A' **+** offset**)))** **{**

encrypt**.**add**((**char**)(**'A' **+** offset**));**

**}**

offset**++;**

**}**

**}**

//大写字母A~Z 对应的加密替换字母

//生成加密字母时需要的偏移量，大小写都使用这个偏移量即可

//即字母A~Z 或者 a~z的加密偏移量

int**[]** encryBook **=** **new** int**[**26**];**

**for** **(**int i**=**0**;** i**<**26**;** i**++)** **{**

encryBook**[**i**]** **=** encrypt**.**get**(**i**)** **-** **(**int**)(**'A' **+** i**);**

**}**

//加密时需要根据字母的下标在加密编码表中查询其对应的偏移量

char**[]** encryptData **=** **new** char**[**data**.**length**()];**

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

int offset2 **=** dataUp**.**charAt**(**i**)-**'A'**;**

char ch **=** data**.**charAt**(**i**);**

//空格不需要加密

**if** **(**ch **!=** ' '**)** **{**

encryptData**[**i**]** **=** **(**char**)** **(**ch**+**encryBook**[**offset2**]);**

**}** **else** **{**

encryptData**[**i**]** **=** ch**;**

**}**

**}**

System**.**out**.**println**(**String**.**valueOf**(**encryptData**));**

**}**

in**.**close**();**

**}**

**}**

# 多线程顺序执行



**import** java**.**util**.**Scanner**;**

public class Main **{**

static char**[]** gWrite **=** **new** char**[**1032**];**

//step:1~4之间循环，控制4个线程顺序执行

//必须使用volatile修饰 否则程序无法安装预想的运行

volatile static int index **=** 0**,** step **=** 1**;**

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

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

**while(**in**.**hasNext**()){**

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

WriteThread thread1 **=** **new** WriteThread**(**'A'**,** n**,** 1**);**

WriteThread thread2 **=** **new** WriteThread**(**'B'**,** n**,** 2**);**

WriteThread thread3 **=** **new** WriteThread**(**'C'**,** n**,** 3**);**

WriteThread thread4 **=** **new** WriteThread**(**'D'**,** n**,** 4**);**

thread1**.**start**();**

thread2**.**start**();**

thread3**.**start**();**

thread4**.**start**();**

Thread**.**sleep**(**1000**);**

System**.**out**.**print**(**String**.**valueOf**(**gWrite**).**trim**());**

**}**

in**.**close**();**

**}**

static synchronized void writeToArray**(**char ch**)** **{**

gWrite**[**index**++]** **=** ch**;**

**}**

static class WriteThread **extends** Thread **{**

char c**;**

int times**;**

int runFlag**;**

public WriteThread**(**char c**,** int times**,** int runFlag**)** **{**

**this.**c **=** c**;**

**this.**times **=** times**;**

**this.**runFlag **=** runFlag**;**

**}**

@Override

public void run**()** **{**

**while** **(**times **>** 0**)** **{**

**if** **(**step **==** runFlag**)** **{**

times**--;**

writeToArray**(**c**);**

step**++;**

**if** **(**step **>** 4**)** step **=** 1**;**

**}**

**}**

**}**

**}**

**}**

**import** java**.**util**.**Scanner**;**

public class Main **{**

static char**[]** gWrite **=** **new** char**[**1032**];**

//step:1~4之间循环，控制4个线程顺序执行

//必须使用volatile修饰 否则程序无法安装预想的运行

volatile static int index **=** 0**,** step **=** 1**,** byeBye **=** 4**;**

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

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

**while(**in**.**hasNext**()){**

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

WriteThread thread1 **=** **new** WriteThread**(**'A'**,** n**,** 1**);**

WriteThread thread2 **=** **new** WriteThread**(**'B'**,** n**,** 2**);**

WriteThread thread3 **=** **new** WriteThread**(**'C'**,** n**,** 3**);**

WriteThread thread4 **=** **new** WriteThread**(**'D'**,** n**,** 4**);**

thread1**.**start**();**

thread2**.**start**();**

thread3**.**start**();**

thread4**.**start**();**

**while** **(**byeBye **!=** 0**);**

System**.**out**.**println**(**String**.**valueOf**(**gWrite**).**trim**());**

**}**

in**.**close**();**

**}**

static synchronized void writeToArray**(**char ch**)** **{**

gWrite**[**index**++]** **=** ch**;**

**}**

static synchronized void sayGoodBye**()** **{**

byeBye**--;**

// System.out.println(byeBye);

**}**

static class WriteThread **extends** Thread **{**

char c**;**

int times**;**

int runFlag**;**

public WriteThread**(**char c**,** int times**,** int runFlag**)** **{**

**this.**c **=** c**;**

**this.**times **=** times**;**

**this.**runFlag **=** runFlag**;**

**}**

@Override

public void run**()** **{**

**while** **(**times **>** 0**)** **{**

**if** **(**step **==** runFlag**)** **{**

times**--;**

writeToArray**(**c**);**

step**++;**

**if** **(**step **>** 4**)** step **=** 1**;**

**}**

**}**

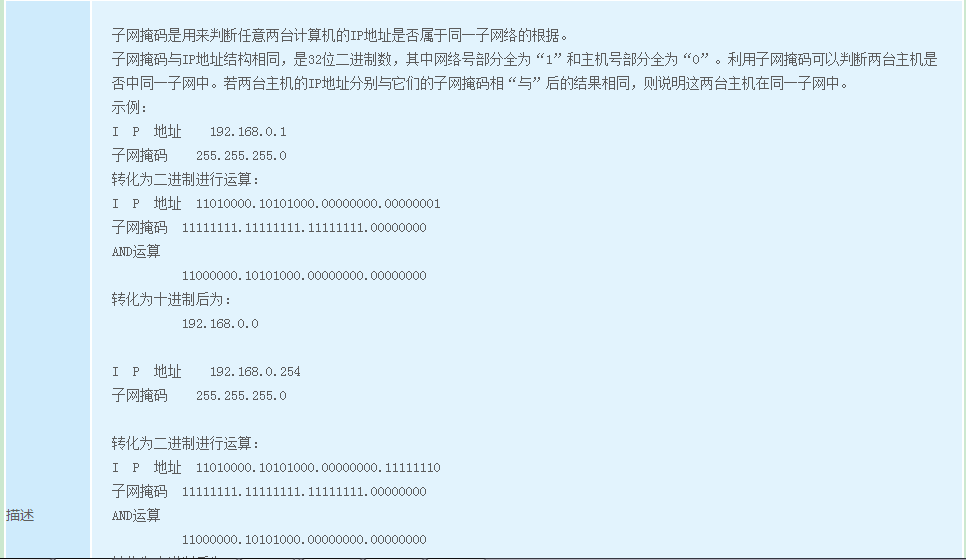
sayGoodBye**();**

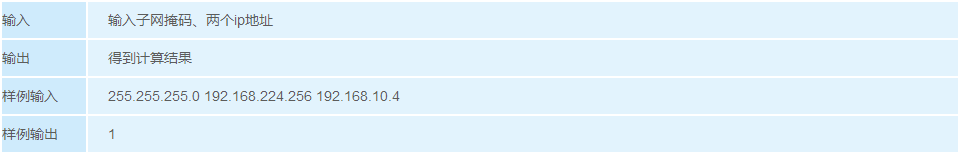
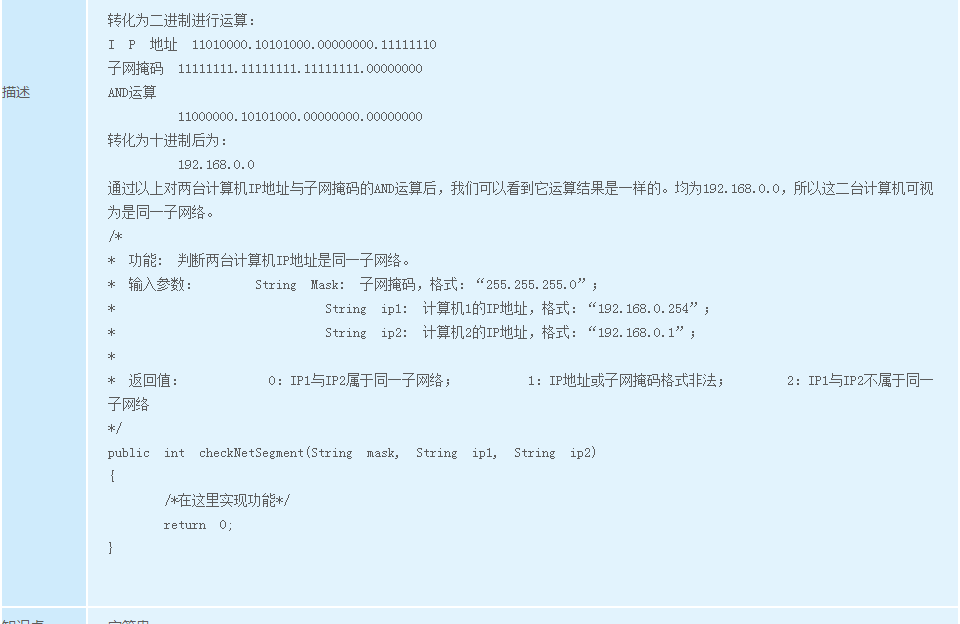
**}**

**}**

**}**

# 判断两个IP是否属于同一子网





package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

// System.out.println(validIp("192.368.0.252"));

**while** **(**scanner**.**hasNext**())** **{**

String mask **=** scanner**.**next**();**

String ip1 **=** scanner**.**next**();**

String ip2 **=** scanner**.**next**();**

List**<**Integer**>** maskList **=** validSubnetMask**(**mask**);**

List**<**Integer**>** ip1List **=** validIp**(**ip1**);**

List**<**Integer**>** ip2List **=** validIp**(**ip2**);**

**if** **(**maskList **!=** **null** **&&** ip1List **!=** **null** **&&** ip2List **!=** **null)** **{**

int i**;**

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

**if** **((**ip1List**.**get**(**i**)&**maskList**.**get**(**i**))** **!=**

**(**ip2List**.**get**(**i**)&**maskList**.**get**(**i**)))** **{**

**break;**

**}**

**}**

**if** **(**i **<** 4**)** **{**

System**.**out**.**println**(**2**);**

**}** **else** **{**

System**.**out**.**println**(**0**);**

**}**

**}** **else** **{**

System**.**out**.**println**(**1**);**

**}**

**}**

**}**

static List**<**Integer**>** validIp**(**String ip**)** **{**

ArrayList**<**Integer**>** iPNum **=** **new** ArrayList**<>();**

String**[]** ms **=** ip**.**split**(**"\\."**);**

**if** **(**ms**.**length **!=** 4**)** **{**

**return** iPNum**;**

**}**

StringBuilder sb **=** **new** StringBuilder**();**

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

int num **=** Integer**.**valueOf**(**ms**[**i**]);**

**if** **(**num **>=** 0 **&&** num **<=** 255**)** **{**

iPNum**.**add**(**num**);**

**}**

**}**

**if** **(**iPNum**.**size**()** **==** 4**)** **return** iPNum**;**

**return** **null;**

**}**

static List**<**Integer**>** validSubnetMask**(**String mask**)** **{**

ArrayList**<**Integer**>** maskNum **=** **new** ArrayList**<>();**

String**[]** ms **=** mask**.**split**(**"\\."**);**

**if** **(**ms**.**length **!=** 4**)** **{**

**return** maskNum**;**

**}**

StringBuilder sb **=** **new** StringBuilder**();**

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

int num **=** Integer**.**valueOf**(**ms**[**i**]);**

**if** **(**num **>=** 0 **&&** num **<=** 255**)** **{**

sb**.**append**(**getBinary**(**num**));**

maskNum**.**add**(**num**);**

**}**

**}**

**if** **(**sb**.**length**()** **==** 32**)** **{**

// System.out.println(sb.toString());

int index **=** sb**.**lastIndexOf**(**"10"**);**

**if** **(!**sb**.**substring**(**0**,** index**+**1**).**contains**(**"0"**))** **{**

//valid

**return** maskNum**;**

**}**

**}**

**return** **null;**

**}**

static String getBinary**(**int num**)** **{**

StringBuilder sb **=** **new** StringBuilder**();**

String bin **=** Integer**.**toBinaryString**(**num**);**

**if** **(**bin**.**length**()** **<** 8**)** **{**

**for** **(**int i**=**0**;** i**<**8**-**bin**.**length**();** i**++)** **{**

sb**.**append**(**"0"**);**

**}**

**}**

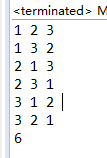
sb**.**append**(**bin**);**

**return** sb**.**toString**();**

**}**

**}**

# 利用dfs求排列组合



public class Main **{**

static int**[]** box **=** **new** int**[**11**];**

static int**[]** book **=** **new** int**[**11**];**

static int n**;**

static long times**=**0**;**

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

n **=** 3**;**//求1~n的全排列组合

dfs**(**1**);**

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

**}**

static void dfs**(**int step**)** **{**

**if** **(**step **==** n**+**1**)** **{**

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

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

**}**

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

times**++;**

**return;**

**}**

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

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

box**[**step**]** **=** i**;**

book**[**i**]** **=** 1**;**

dfs**(**step**+**1**);**

book**[**i**]** **=** 0**;**

**}**

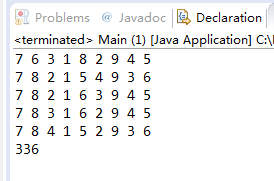
**}**

**return;**

**}**

**}**

问题延伸，对编号1~9的扑克牌，放到9个盒子里，使得□□□+□□□=□□□。



package com**.**oj**;**

public class Main **{**

static int**[]** box **=** **new** int**[**11**];**

static int**[]** book **=** **new** int**[**11**];**

static int n**;**

static long times**=**0**;**

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

n **=** 9**;**//求1~n的全排列组合

dfs**(**1**);**

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

**}**

static void dfs**(**int step**)** **{**

**if** **(**step **==** n**+**1**)** **{**

**if** **(**validEquality**(**box**))** **{**

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

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

**}**

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

times**++;**

**}**

**return;**

**}**

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

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

box**[**step**]** **=** i**;**

book**[**i**]** **=** 1**;**

dfs**(**step**+**1**);**

book**[**i**]** **=** 0**;**

**}**

**}**

**return;**

**}**

static boolean validEquality**(**int**[]** ns**)** **{**

**if** **(**getNum**(**ns**,** 1**)+**getNum**(**ns**,** 4**)** **==** getNum**(**ns**,** 7**))** **{**

**return** **true;**

**}**

**return** **false;**

**}**

static int getNum**(**int**[]** ns**,**int i**)** **{**

**return** ns**[**i**]\***100**+**ns**[**i**+**1**]\***10**+**ns**[**i**+**2**];**

**}**

**}**

# dfs走迷宫

**import** java**.**util**.**Scanner**;**

public class Main **{**

/\* 输入数据：行数 列数 迷宫组成 起始位置 目标位置

6 5

0 0 1 0 0

0 0 0 0 1

0 0 1 0 1

0 1 0 0 0

0 0 0 1 1

1 0 0 0 0

1 1 6 5

\*/

static int row**,**column**,**startX**,**startY**,**endX**,**endY**,**steps**=**1**;**

static int**[][]** maze**,**book**;**

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

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

row **=** scanner**.**nextInt**();**

column **=** scanner**.**nextInt**();**

maze **=** **new** int**[**row**+**1**][**column**+**1**];**

book **=** **new** int**[**row**+**1**][**column**+**1**];**

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

**for** **(**int j**=**1**;** j**<=**column**;** j**++)** **{**

maze**[**i**][**j**]** **=** scanner**.**nextInt**();**

**}**

**}**

startX **=** scanner**.**nextInt**();**

startY **=** scanner**.**nextInt**();**

endX **=** scanner**.**nextInt**();**

endY **=** scanner**.**nextInt**();**

initBook**();**

dfs**(**startX**,** startY**,** 1**);**

System**.**out**.**println**(**"共有 "**+**totalMethod**+**" 种走法"**);**

System**.**out**.**println**(**"最少步数是 "**+**leastTimes**);**

**}**

static int leastTimes **=** Integer**.**MAX\_VALUE**,**totalMethod**=**0**;**

static void dfs**(**int x**,** int y**,** int step**)** **{**

int**[][]** next **=** **{{**0**,**1**},{**1**,**0**},{**0**,-**1**},{-**1**,**0**}};**

**if** **(**x **==** startX **&&** y **==** startY**)** **{**

book**[**x**][**y**]** **=** **-**1**;**//标记起始位置

**}**

**if** **(**x **==** endX **&&** y **==** endY**)** **{**

System**.**out**.**println**(**step**-**1**);**//走了几步

printMaze**(**book**);**

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

**if** **(**step**-**1 **<** leastTimes**)** **{**

leastTimes **=** step **-** 1**;**

**}**

totalMethod**++;**

**return;**

**}**

int tx **=** 0**,**ty **=** 0**;**

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

tx **=** x **+** next**[**i**][**0**];**

ty **=** y **+** next**[**i**][**1**];**

//tx:1~row ty:1~column

**if** **(**tx**<** 1 **||** ty**<**1 **||** tx**>**row **||** ty**>**column**)** **{**

**continue;**

**}**

**if** **(**maze**[**tx**][**ty**]** **==** 0 **&&** book**[**tx**][**ty**]** **==** 0**)** **{**

book**[**tx**][**ty**]** **=** step**;**

dfs**(**tx**,** ty**,** step**+**1**);**

book**[**tx**][**ty**]** **=** 0**;**

**}**

**}**

**if** **(**x **==** startX **&&** y **==** startY**)** **{**

book**[**x**][**y**]** **=** 0**;**

**}**

**}**

static void initBook**()** **{**

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

**for** **(**int j**=**1**;** j**<**maze**[**0**].**length**;** j**++)** **{**

**if** **(**maze**[**i**][**j**]** **==** 1**)** **{**

book**[**i**][**j**]** **=** 44**;**//表示一堵墙

**}**

**}**

**}**

**}**

static void printMaze**(**int**[][]** maze**)** **{**

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

**for** **(**int j**=**1**;** j**<**maze**[**0**].**length**;** j**++)** **{**

System**.**out**.**printf**(**"%-3d"**,**maze**[**i**][**j**]);**

**}**

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

**}**

**}**

**}**

**另一种方式**

package com**.**openhome**;**

public class Demo **{**

/\*\*

\* @param args

\*/

static int endx **=** 6**,** endy **=** 5**,** startx **=** 1**,** starty **=** 0**;**

static int**[][]** maze**=** **{**

**{**2**,**2**,**2**,**2**,**2**,**2**,**2**},**

**{**0**,**0**,**0**,**0**,**0**,**0**,**2**},**

**{**2**,**0**,**2**,**2**,**2**,**0**,**2**},**

**{**2**,**0**,**0**,**0**,**0**,**0**,**2**},**

**{**2**,**2**,**0**,**2**,**0**,**2**,**2**},**

**{**2**,**0**,**0**,**0**,**0**,**0**,**2**},**

**{**2**,**2**,**2**,**2**,**2**,**0**,**2**}**

**};**

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

// TODO Auto-generated method stub

printMaze**();**

System**.**out**.**println**(**visits**(**startx**,** starty**));**

System**.**out**.**println**(**"judgeTimes:"**+** judgeTimes**);**

**}**

// static int endx = 2, endy = 1, startx = 1, starty = 0;

// static int[][] maze= {

// {2,2,2},

// {0,0,2},

// {2,0,2}

// };

static boolean success **=** **false;**

static int judgeTimes**=**0**;**//判断次数

static int steps**=**0**;**//走出迷宫步数

//只计算出一条路径 然后结束

public static boolean visit**(**int x**,** int y**)** **{**

judgeTimes**++;**

//标志已经走过的路段

maze**[**x**][**y**]** **=** 1**;**

**if** **(**x**==**endx **&&** y**==**endy**)** **{**

success **=** **true;**

printMaze**();**

**}**

//right

**if** **(!**success **&&** maze**[**x**][**y**+**1**]** **==** 0**)** **{**

visit**(**x**,** y**+**1**);**

**}**

//down

**if** **(!**success **&&** maze**[**x**+**1**][**y**]** **==** 0**)** **{**

visit**(**x**+**1**,** y**);**

**}**

//left

**if** **(!**success **&&** y**-**1**>=**0 **&&** maze**[**x**][**y**-**1**]** **==** 0**)** **{**

visit**(**x**,** y**-**1**);**

**}**

//up

**if** **(!**success **&&** x**-**1**>=**0 **&&** maze**[**x**-**1**][**y**]** **==** 0**)** **{**

visit**(**x**-**1**,** y**);**

**}**

//此路不通 清除标志

**if** **(!**success**)** **{**

maze**[**x**][**y**]** **=** 0**;**

**}**

**return** success**;**

**}**

static int paths **=** 0**;**

//计算出多条路径 返回路径条数

public static int visits**(**int x**,** int y**)** **{**

judgeTimes**++;**

maze**[**x**][**y**]** **=** 1**;**

**if** **(**x**==**endx **&&** y**==**endy**)** **{**

// success = true;

printMaze**();**

paths**++;**

**}**

//right

**if** **(**y**+**1**<**maze**[**x**].**length **&&** maze**[**x**][**y**+**1**]** **==** 0**)** **{**

visits**(**x**,** y**+**1**);**

**}**

//down

**if** **(**x**+**1**<**maze**.**length **&&** maze**[**x**+**1**][**y**]** **==** 0**)** **{**

visits**(**x**+**1**,** y**);**

**}**

//left

**if** **(**y**-**1**>=**0 **&&** maze**[**x**][**y**-**1**]** **==** 0**)** **{**

visits**(**x**,** y**-**1**);**

**}**

//up

**if** **(**x**-**1**>=**0 **&&** maze**[**x**-**1**][**y**]** **==** 0**)** **{**

visits**(**x**-**1**,** y**);**

**}**

**if** **(!**success**)** **{**

maze**[**x**][**y**]** **=** 0**;**

**}**

**return** paths**;**

**}**

static void printMaze**()** **{**

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

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

// System.out.printf(" %d ",maze[i][j]);

**if** **(**maze**[**i**][**j**]** **==** 2**)** **{**

System**.**out**.**print**(**"■ "**);**

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

steps**++;**

System**.**out**.**print**(**"♂ "**);**

**}else** **{**

System**.**out**.**print**(**"○ "**);**

**}**

**}**

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

**}**

System**.**out**.**println**(**"steps:"**+**steps**);**

steps **=** 0**;**

**}**

**}**

# 利用bfs进行迷宫搜索

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Scanner**;**

**import** javax**.**xml**.**soap**.**Node**;**

public class Main **{**

/\* 输入数据：行数 列数 迷宫组成 起始位置 目标位置

6 5

0 0 1 0 0

0 0 0 0 1

0 0 1 0 1

0 1 0 0 0

0 0 0 1 1

1 0 0 0 0

1 1 6 5

\*/

static int row**,**column**,**startX**,**startY**,**endX**,**endY**,**steps**=**1**;**

static int**[][]** maze**,**book**;**

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

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

row **=** scanner**.**nextInt**();**

column **=** scanner**.**nextInt**();**

maze **=** **new** int**[**row**+**1**][**column**+**1**];**

book **=** **new** int**[**row**+**1**][**column**+**1**];**

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

**for** **(**int j**=**1**;** j**<=**column**;** j**++)** **{**

maze**[**i**][**j**]** **=** scanner**.**nextInt**();**

**}**

**}**

startX **=** scanner**.**nextInt**();**

startY **=** scanner**.**nextInt**();**

endX **=** scanner**.**nextInt**();**

endY **=** scanner**.**nextInt**();**

initBook**();**

bfs**();**

**}**

static int leastTimes **=** Integer**.**MAX\_VALUE**,**totalMethod**=**0**;**

static void bfs**()** **{**

int**[][]** next **=** **{{**0**,**1**},{**1**,**0**},{**0**,-**1**},{-**1**,**0**}};**

ArrayList**<**Note**>** que **=** **new** ArrayList**<>();**

int head**=**1**,**tail**=**1**;**

Note note **=** **new** Note**();**

//队列初始化

que**.**add**(**note**);**

que**.**add**(**note**);**//坐标到1

que**.**get**(**tail**).**x **=** startX**;**

que**.**get**(**tail**).**y **=** startY**;**

que**.**get**(**tail**).**ft **=** 0**;**

que**.**get**(**tail**).**st **=** 0**;**

tail**++;**

book**[**startX**][**startY**]** **=** 1**;**

int endFlag **=** 0**;**

int tx **=** 0**,**ty **=** 0**;**

**while** **(**head **<** tail**)** **{**

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

//根据队列首部的点 判断下一波可以到达的点 并加入到队列尾部

tx **=** que**.**get**(**head**).**x **+** next**[**i**][**0**];**

ty **=** que**.**get**(**head**).**y **+** next**[**i**][**1**];**

//tx:1~row ty:1~column

**if** **(**tx**<** 1 **||** ty**<**1 **||** tx**>**row **||** ty**>**column**)** **{**

**continue;**

**}**

**if** **(**maze**[**tx**][**ty**]** **==** 0 **&&** book**[**tx**][**ty**]** **==** 0**)** **{**

book**[**tx**][**ty**]** **=** 1**;**

//将能到达的点加入队列尾部

Note note2 **=** **new** Note**();**

que**.**add**(**note2**);**

que**.**get**(**tail**).**x **=** tx**;**

que**.**get**(**tail**).**y **=** ty**;**

que**.**get**(**tail**).**ft **=** head**;**

que**.**get**(**tail**).**st **=** que**.**get**(**head**).**st**+**1**;**

tail**++;**

// System.out.println(tail);

**}**

//判断是否到达目标点

**if** **(**tx **==** endX **&&** ty **==** endY**)** **{**

endFlag **=** 1**;**

**break;**

**}**

**}**

**if** **(**endFlag **==** 1**)** **break;**

//将已经检查过所有可到达位置的点从队列移除

head**++;**

**}**

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

System**.**out**.**println**(**que**.**get**(**tail**-**1**).**st**);**

// for (Note n : que) {

// System.out.println(n);

// }

**}**

static void initBook**()** **{**

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

**for** **(**int j**=**1**;** j**<**maze**[**0**].**length**;** j**++)** **{**

**if** **(**maze**[**i**][**j**]** **==** 1**)** **{**

book**[**i**][**j**]** **=** 44**;**//表示一堵墙

**}**

**}**

**}**

**}**

static void printMaze**(**int**[][]** maze**)** **{**

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

**for** **(**int j**=**1**;** j**<**maze**[**0**].**length**;** j**++)** **{**

System**.**out**.**printf**(**"%-3d"**,**maze**[**i**][**j**]);**

**}**

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

**}**

**}**

**}**

class Note **{**

int x**;**

int y**;**

int ft**;**

int st**;**

@Override

public String toString**()** **{**

**return** x**+**" "**+**y**+**" "**+**ft**+**" "**+**st**;**

**}**

**}**

方法2：

public class Main **{**

static double PRECISION **=** 0.0001**;**

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

//1,1->5,4

int**[][]** map **=** **{{**0**,**0**,**0**,**0**,**0**},**

**{**0**,**0**,**0**,**1**,**0**},**//1~4

**{**0**,**0**,**0**,**0**,**0**},**

**{**0**,**0**,**0**,**1**,**0**},**

**{**0**,**1**,**1**,**0**,**0**},**

**{**0**,**0**,**0**,**0**,**1**}};**//

int Xborder **=** map**.**length**-**1**;**

int Yborder **=** map**[**0**].**length**-**1**;**

int**[][]** book **=** **new** int**[**Xborder**+**1**][**Yborder**+**1**];**

MyQueue queue **=** **new** MyQueue**();**

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

int startX **=** 1**;**

int startY **=** 1**;**

int endX **=** 4**;**

int endY **=** 3**;**

queue**.**add**(**startX**,** startY**,** 0**);**

book**[**startX**][**startY**]** **=** 1**;**

boolean endFlag **=** **false;**

**while** **(!**queue**.**isEmputy**())** **{**

int**[][]** next **=** **{{**0**,**1**},{**0**,-**1**},{**1**,**0**},{-**1**,**0**}};**

int**[]** node **=** queue**.**get**();**

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

int toX **=** node**[**0**]** **+** next**[**i**][**0**];**

int toY **=** node**[**1**]** **+** next**[**i**][**1**];**

**if** **(**toX **<** 1 **||** toX **>** Xborder **||** toY **<**1 **||** toY **>** Yborder**)** **{**

**continue;**

**}**

**if** **(**map**[**toX**][**toY**]** **!=** 1 **&&** book**[**toX**][**toY**]** **==** 0**)** **{**

int step **=** node**[**2**]** **+** 1**;**

queue**.**add**(**toX**,** toY**,** step**);**

book**[**toX**][**toY**]** **=** 1**;**

//注意，如果不加下面这一部分，那么程序会将整个数组遍历一遍的

**if** **(**toX **==** endX **&&** toY **==** endY**)** **{**

endFlag **=** **true;**

**break;**

**}**

**}**

**}**

**if** **(**endFlag**)** **break;**

**}**

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

queue**.**showLast**();**

printArray2**(**book**);**

**}**

static void printArray2**(**int**[][]** a**)** **{**

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

**for** **(**int j**=**1**;** j**<**a**[**0**].**length**;** j**++)** **{**

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

**}**

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

**}**

**}**

**}**

class MyQueue **{**

private static int MAXSIZE **=** 10**;**

private int**[][]** data**;**

private int head**;**

private int tail**;**

private int size**;**

public MyQueue**()** **{**

data **=** **new** int**[**MAXSIZE**][**3**];**

head **=** 0**;**

tail **=** 0**;**

**}**

public boolean isFull**()** **{**

**if** **((**tail**+**1**)** **%** MAXSIZE **==** head**)**

**return** **true;**

**else**

**return** **false;**

**}**

public boolean isEmputy**()** **{**

**if** **(**tail **==** head**)**

**return** **true;**

**else**

**return** **false;**

**}**

public int getSize**()** **{**

size **=** **(**tail **-** head **+** MAXSIZE**)** **%** MAXSIZE**;**

**return** size**;**

**}**

public boolean add**(**int x**,** int y**,** int s**)** **{**

**if** **(**isFull**())** **{**

**return** **false;**

**}** **else** **{**

data**[**tail**][**0**]** **=** x**;**

data**[**tail**][**1**]** **=** y**;**

data**[**tail**][**2**]** **=** s**;**

tail **=** **(**tail**+**1**)** **%** MAXSIZE**;**

**return** **true;**

**}**

**}**

public int**[]** get**()** **{**

**if** **(**isEmputy**())** **{**

**return** **null;**

**}** **else** **{**

int**[]** e **=** data**[**head**];**

head **=** **(**head**+**1**)** **%** MAXSIZE**;**

**return** e**;**

**}**

**}**

@Override

public String toString**()** **{**

StringBuilder sb **=** **new** StringBuilder**();**

sb**.**append**(**"h:"**+**head**+**" t:"**+**tail**+**" s:"**+**getSize**()+**"\n"**);**

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

sb**.**append**(**data**[**i**][**0**]+**"\_"**+**data**[**i**][**1**]+**"\_"**+**data**[**i**][**2**]+**" "**);**

**}**

// sb.append("\n");

**return** sb**.**toString**();**

**}**

public void showLast**()** **{**

int i **=** **(**tail **-** 1**)** **%** MAXSIZE**;**

System**.**out**.**println**(**data**[**i**][**0**]+**"\_"**+**data**[**i**][**1**]+**"\_"**+**data**[**i**][**2**]);**

**}**

**}**

# 单链表实现

public class Main **{**

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

MyLink link **=** **new** MyLink**();**

link**.**append**(**11**);**

link**.**append**(**22**);**

link**.**append**(**33**);**

link**.**append**(**44**);**

link**.**append**(**55**);**

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

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

System**.**out**.**println**(**link**.**get**(**5**));**

**}**

**}**

class MyLink **{**

//头节点指向第一个节点

class Node **{**

Object o**;**

Node next **=** **null;**

**}**

public Node head**;**

public int size**;**

public MyLink**()** **{**

head **=** **new** Node**();**

size **=** 0**;**

**}**

//在链表的最后位置插入一个节点

public void append**(**Object o**)** **{**

Node node **=** **new** Node**();**

node**.**o **=** o**;**

**if** **(**head**.**next **==** **null)** **{**

head**.**next **=** node**;**

**}** **else** **{**

Node iterator **=** head**;**

**while** **(**iterator**.**next **!=** **null)** **{**

iterator **=** iterator**.**next**;**

**}**

iterator**.**next **=** node**;**

**}**

size**++;**

**}**

//获取第i个节点的数据,没有则为null

public Object get**(**int i**)** **{**

Object res **=** **null;**

Node iterator **=** head**;**

int index **=** 0**;**

**while** **(**iterator **!=** **null** **&&** index **<** i**)** **{**

iterator **=** iterator**.**next**;**

index**++;**

**}**

**if** **(**iterator **!=** **null)** **{**

res **=** iterator**.**o**;**

**}**

**return** res**;**

**}**

//在第i个节点处插入 o

//i=0和i=1，都是在第一个节点插入数据

//成功返回true

public boolean insert**(**int i**,** Object o**)** **{**

Node iterator **=** head**;**

int index **=** 0**;**

**while** **(**iterator **!=** **null** **&&** index **<** i**-**1**)** **{**

iterator **=** iterator**.**next**;**

index**++;**

**}**

**if** **(**iterator **!=** **null)** **{**

Node node **=** **new** Node**();**

node**.**o **=** o**;**

node**.**next **=** iterator**.**next**;**

iterator**.**next **=** node**;**

size**++;**

**return** **true;**

**}**

**return** **false;**

**}**

//删除第i个节点的数据，并返回删除的数据，删除失败返回null

public Object delete**(**int i**)** **{**

Object returnObject **=** **null;**

Node iterator **=** head**;**

int index **=** 0**;**

**while** **(**iterator**.**next **!=** **null** **&&** index **<** i**-**1**)** **{**

iterator **=** iterator**.**next**;**

index**++;**

**}**

**if** **(**iterator**.**next **!=** **null)** **{**

returnObject **=** iterator**.**next**.**o**;**

iterator**.**next **=** iterator**.**next**.**next**;**

size**--;**

**}**

**return** returnObject**;**

**}**

public String toString**()** **{**

Node iterator **=** head**;**

//指向第一个节点

StringBuilder sb **=** **new** StringBuilder**();**

sb**.**append**(**"size:"**+**size**+**" {"**);**

**while** **(**iterator**.**next **!=** **null)** **{**

iterator **=** iterator**.**next**;**

sb**.**append**(**iterator**.**o**+**" "**);**

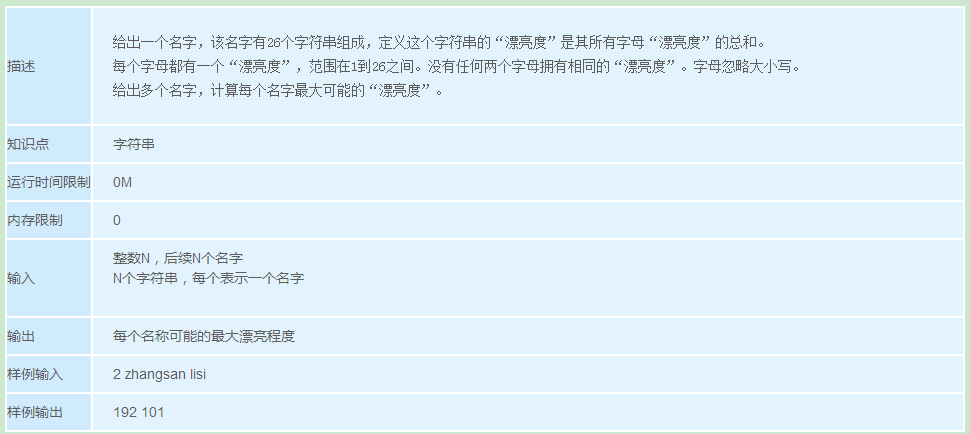
**}**

**return** sb**.**deleteCharAt**(**sb**.**length**()-**1**).**append**(**"}"**).**toString**();**

**}**

**}**

# 名字的漂亮度



**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Arrays**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

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

ArrayList**<**String**>** names **=** **new** ArrayList**<>();**

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

names**.**add**(**scanner**.**next**());**

**}**

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

System**.**out**.**println**(**getLongest**(**names**.**get**(**i**)));**

**}**

**}**

**}**

static int getLongest**(**String name**)** **{**

ArrayList**<**Character**>** nch **=** **new** ArrayList**<>();**

int**[]** ts **=** **new** int**[**name**.**length**()];**

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

Character ch **=** name**.**charAt**(**i**);**

**if** **(!**nch**.**contains**(**ch**))** **{**

nch**.**add**(**ch**);**

int index **=** nch**.**indexOf**(**ch**);**

ts**[**index**]++;**

**}** **else** **{**

int index **=** nch**.**indexOf**(**ch**);**

ts**[**index**]++;**

**}**

**}**

int sum **=** 0**;**

int p **=** 26**;**

Arrays**.**sort**(**ts**);**

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

// System.out.print(ts[i]+" ");

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

sum **+=** ts**[**i**]\***p**;**

p**--;**

**}**

**}**

**return** sum**;**

**}**

**}**

# 蛇行矩阵



**import** java**.**util**.**Scanner**;**

public class Main **{**

static int**[][]** table**;**

static int num **,**index **=** 1**;**

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

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

num **=** scanner**.**nextInt**();**

table **=** **new** int**[**num**][**num**];**

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

fillNum**(**i**);**

**}**

printTable**();**

**}**

static void printTable**()** **{**

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

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

**if** **(**table**[**i**][**j**]** **!=** 0**)** **{**

**if** **(**j **!=** **(**num**-**i**-**1**))//每一行的最后不能加空格**

System**.**out**.**printf**(**"%d "**,**table**[**i**][**j**]);**

**else**

System**.**out**.**printf**(**"%d"**,**table**[**i**][**j**]);**

**}**

**}**

**if** **(**table**[**i**][**0**]** **!=** 0**)**

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

**}**

**}**

//对蛇形矩阵的一个斜行填充数据

static void fillNum**(**int in**)** **{**

int n **=** in**-**1**;**

int x**,**y**=**0**;**

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

x **=** n**-**i**;**

y **=** i**;**

table**[**x**][**y**]** **=** index**++;**

**}**

**}**

**}**

# 利用DP求最长公共子序列

**import** java**.**util**.**Scanner**;**

public class Demo **{**

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

String X **=** "ABCBDABE"**;**

String Y **=** "BDCABAE"**;**

System**.**out**.**println**(**lcs2**(**X**,** Y**,** X**.**length**(),** Y**.**length**()));**

**}**

//Return the length of LCS for X[0...m-1] and Y[0...n-1]

static int lcs**(**String X**,** String Y**,** int m**,** int n**)**

**{**

**if** **(**m **==** 0 **||** n **==** 0**)**

**return** 0**;**

**if** **(**X**.**charAt**(**m**-**1**)** **==** Y**.**charAt**(**n**-**1**))**

**return** lcs**(**X**,** Y**,** m**-**1**,** n**-**1**)** **+** 1**;**

**else**

**return** max**(**lcs**(**X**,** Y**,** m**,** n**-**1**),** lcs**(**X**,** Y**,** m**-**1**,** n**));**

**}**

static int max**(**int a**,** int b**)**

**{**

**return** **(**a**>**b**)?** a**:**b**;**

**}**

/\*\*

\* 返回X[0...m-1]和Y[0...n-1]的LCS的长度

\*/

static int lcs2**(**String X**,** String Y**,** int m**,** int n**)**

**{**

// 动态规划表，大小(m+1)\*(n+1)

int**[][]** table **=** **new** int**[**m**+**1**][**n**+**1**];**

**for(**int i**=**0**;** i**<**m**+**1**;** **++**i**)**

**{**

**for(**int j**=**0**;** j**<**n**+**1**;** **++**j**)**

**{**

// 第一行和第一列置0

**if** **(**i **==** 0 **||** j **==** 0**)**

table**[**i**][**j**]** **=** 0**;**

**else** **if(**X**.**charAt**(**i**-**1**)** **==** Y**.**charAt**(**j**-**1**))**

table**[**i**][**j**]** **=** table**[**i**-**1**][**j**-**1**]** **+** 1**;**

**else**

table**[**i**][**j**]** **=** max**(**table**[**i**-**1**][**j**],** table**[**i**][**j**-**1**]);**

**}**

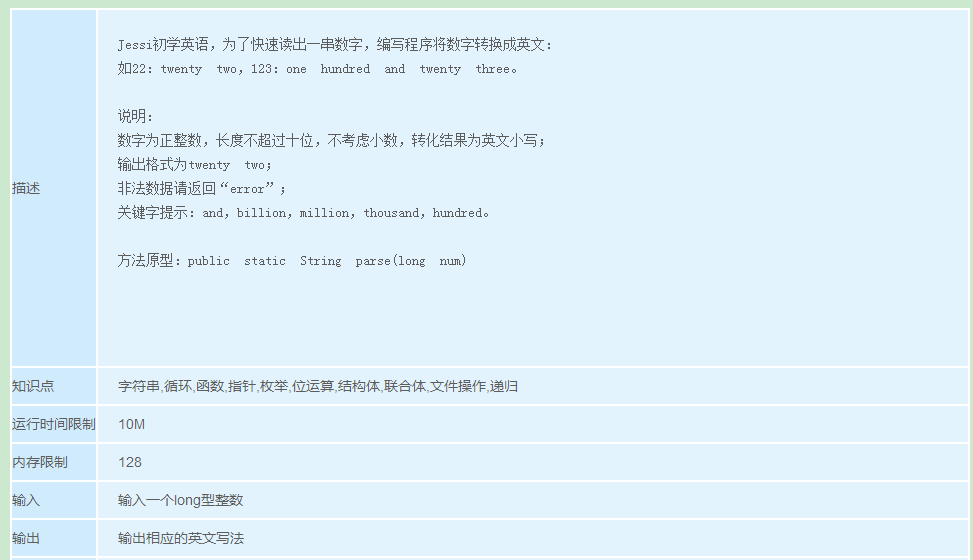
**}**

**return** table**[**m**][**n**];**

**}**

**}**

# 学英语



**import** java**.**util**.**Scanner**;**

public class Main **{**

static String**[]** table1 **=** **{**""**,**"one"**,**"two"**,**"three"**,**"four"**,**"five"**,**//1~5

"six"**,**"seven"**,**"eight"**,**"nine"**,**"ten"**,**//6~10

"eleven"**,**"twelve"**,**"thirteen"**,**"fourteen"**,**"fifteen"**,**//11~15

"sixteen"**,**"seventeen"**,**"eighteen"**,**"nineteen"**};**//16~19

static String**[]** table2 **=** **{**""**,**""**,**"twenty"**,**"thirty"**,**"forty"**,**"fifty"**,**"sixty"**,**

"seventy"**,**"eighty"**,**"ninety"**};**

static String**[]** table3 **=** **{**""**,**" thousand "**,**" million "**,**" billion "**};**

static String and **=** " and "**;**

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

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

**while** **(**scanner**.**hasNext**())** **{**

**try** **{**

long num **=** scanner**.**nextLong**();**

String res = *parse*(num);

res = res.replace("\\s+", " ");

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

**}** **catch** **(**Exception e**)** **{**

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

scanner**.**next**();**

**}**

**}**

scanner**.**close**();**

**}**

private static String parseNum**(**long num**)** **{**

String strNum **=** String**.**valueOf**(**num**);**

int len **=** strNum**.**length**();**

//将数字从最低位往最高位方向每3位分组，分成n组

int n **=** len**/**3 **+** **(**len**%**3 **==** 0 **?** 0 **:** 1**);**

//分组结果存储在ss中

//如num = 1234567897;则ss[0]="897"

//ss[1]="567",ss[2]="234",ss[3]="1"

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

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

**if** **(**len**-**3 **<** 0**)** **{**

ss**[**i**]** **=** strNum**.**substring**(**0**,** len**);**

**}** **else** **{**

ss**[**i**]** **=** strNum**.**substring**(**len**-**3**,** len**);**

len **-=** 3**;**

**}**

**}**

// for (String s : ss) {

// System.out.println(s);

// }

StringBuilder sb **=** **new** StringBuilder**();**

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

sb**.**insert**(**0**,** parse3Num**(**ss**[**i**])+**table3**[**i**]);**

**}**

**return** sb**.**toString**().**trim**();**

**}**

//将3位数以内的数字解析成英文

private static String parse3Num**(**int num**)** **{**

**return** parse3Num**(**String**.**valueOf**(**num**));**

**}**

private static String parse3Num**(**String strNum**)** **{**

StringBuilder sbNum **=** **new** StringBuilder**();**

int lastId **=** strNum**.**length**()-**1**;**

**if** **(**lastId **==** 0**)** **{**

int ge **=** Integer**.**valueOf**(**strNum**.**substring**(**lastId**,** lastId**+**1**));**

sbNum**.**append**(**table1**[**ge**]);**

**}**

int ge **=** 0**,**shi**=**0**;**

**if** **(**lastId **>** 0**)** **{**

//提取最后个位和十位的值

**if** **(**strNum**.**charAt**(**lastId**-**1**)** **==** '1'**)** **{**

int geShi **=** Integer**.**valueOf**(**strNum**.**substring**(**lastId**-**1**,** lastId**+**1**));**

sbNum**.**append**(**table1**[**geShi**]);**

ge **=** geShi**%**10**;**

shi **=** geShi**/**10**;**

**}** **else** **{**

ge **=** Integer**.**valueOf**(**strNum**.**substring**(**lastId**,** lastId**+**1**));**

sbNum**.**append**(**table1**[**ge**]);**

shi **=** Integer**.**valueOf**(**strNum**.**substring**(**lastId**-**1**,** lastId**));**

sbNum**.**insert**(**0**,** table2**[**shi**]+**" "**);**

**}**

**}**

**if** **(**lastId **>=** 2**)** **{**

//提取百位数值

int bai **=** Integer**.**valueOf**(**strNum**.**substring**(**lastId**-**2**,** lastId**-**1**));**

**if** **(**bai **!=** 0**)** **{**

String isAnd **=** **(**ge**==**0 **&&** shi**==**0**)?**""**:**and**;**//个位十位为0不加and

sbNum**.**insert**(**0**,** table1**[**bai**]+**" "**+**"hundred"**+**isAnd**);**

**}**

**}**

**return** sbNum**.**toString**().**trim**();**

**}**

**}**

**import** java**.**util**.**Scanner**;**

public class Main **{**

static String**[]** table1 **=** **{**""**,**"one"**,**"two"**,**"three"**,**"four"**,**"five"**,**//1~5

"six"**,**"seven"**,**"eight"**,**"nine"**,**"ten"**,**//6~10

"eleven"**,**"twelve"**,**"thirteen"**,**"fourteen"**,**"fifteen"**,**//11~15

"sixteen"**,**"seventeen"**,**"eighteen"**,**"nineteen"**};**//16~19

static String**[]** table2 **=** **{**""**,**""**,**"twenty"**,**"thirty"**,**"forty"**,**"fifty"**,**"sixty"**,**

"seventy"**,**"eighty"**,**"ninety"**};**

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

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

**while** **(**scanner**.**hasNext**())** **{**

**try** **{**

long num **=** scanner**.**nextLong**();**

System**.**out**.**println**(**parse**(**num**));**

**}** **catch** **(**Exception e**)** **{**

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

scanner**.**next**();**

**}**

**}**

scanner**.**close**();**

**}**

private static String parse**(**long num**)** **{**

**if** **(**num **<** 0**)** **{**

**return** "error"**;**

**}** **else** **if** **(**num **<** 20**)** **{**

**return** table1**[(**int**)** num**];**

**}** **else** **if** **(**num **<** 100**)** **{**

**if** **(**num **%** 10 **==** 0**)**

**return** table2**[(**int**)** **(**num**/**10**)];**

**else**

**return** table2**[(**int**)** **(**num**/**10**)]+**" "**+** parse**(**num**%**10**);**

**}** **else** **if** **(**num **<** 1000**)** **{**

**if** **(**num **%** 100 **==** 0**)**

**return** table1**[(**int**)** **(**num**/**100**)]+**" hundred"**;**

**else**

**return** table1**[(**int**)** **(**num**/**100**)]+**" hundred and "**+**parse**(**num**%**100**);**

**}** **else** **if** **(**num **<** 1000\_000**)** **{**

**if** **(**num **%** 1000 **==** 0**)**

**return** parse**(**num**/**1000**)+**" thousand"**;**

**else**

**return** parse**(**num**/**1000**)+**" thousand " **+** parse**(**num**%**1000**);**

**}** **else** **if** **(**num **<** 1000\_000\_000L**)** **{**

**if** **(**num **%** 1000\_000 **==** 0**)**

**return** parse**(**num**/**1000\_000**)+**" million"**;**

**else**

**return** parse**(**num**/**1000\_000**)+**" million " **+** parse**(**num**%**1000\_000**);**

**}** **else** **if** **(**num **<** 10\_000\_000\_000L**)** **{**

**if** **(**num **%** 1000\_000\_000L **==** 0**)**

**return** parse**(**num**/**1000\_000\_000L**)+**" billion"**;**

**else**

**return** parse**(**num**/**1000\_000\_000L**)+**" billion " **+** parse**(**num**%**1000\_000\_000L**);**

**}** **else** **{**

**return** "error"**;**

**}**

**}**

**}**

# 二叉树

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

public class Demo **{**

static List**<**Integer**>** tree **=** **new** ArrayList**<>();**

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

tree**.**add**(**0**);**

**for** **(**int i**=**4**;**i**>=**1**;**i**--)** **{**

tree**.**add**(**i**);**

**}**

**for** **(**int i**=**8**;**i**<=**12**;**i**++)** **{**

tree**.**add**(**i**);**

**}**

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

int k **=** tree**.**size**();**

**for** **(**int i**=((**k**-**1**)/**2**);** i**>=**1**;** i**--)** **{**

siftDown**(**i**,** tree**);**

**}**

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

**}**

//节点下移

static void siftDown**(**int i**,** List**<**Integer**>** tree**)** **{**

int len **=** tree**.**size**()-**1**;**//1~len

int RootType **=** 1**;** //1根节点最小 -1：根节点最大

int min**;**

**while** **(**2**\***i **<=** len**)** **{**

//大于左子节点

**if** **(**tree**.**get**(**i**).**compareTo**(**tree**.**get**(**2**\***i**))** **==** RootType**)** **{**

min **=** 2**\***i**;** //记录最小那个节点编号

**}** **else** **{**

min **=** i**;**

**}**

//如果有右子节点 比较

**if** **((**2**\***i**+**1**)** **<=** len **&&** tree**.**get**(**min**).**compareTo**(**tree**.**get**(**i**\***2**+**1**))** **==** RootType**)** **{**

min **=** 2**\***i **+** 1**;**

**}**

//父节点i不是最小节点，交换值，进行下一轮比较

**if** **(**i **!=** min**)** **{**

int temp **=** tree**.**get**(**i**);**

tree**.**set**(**i**,** tree**.**get**(**min**));**

tree**.**set**(**min**,** temp**);**

//从交换后的子节点进行下一轮比较

i **=** min**;**

**}** **else** **{**//父节点就是最小节点了，无需下移，结束

**break;**//退出判断

**}**

**}**

**}**

//往树中插入一个节点,节点上移

static void siftUp**(**Integer val**,** List**<**Integer**>** tree**)** **{**

int RootType **=** **-**1**;**//根节点最小（1：根节点最大）

tree**.**add**(**val**);**

int i **=** tree**.**size**()-**1**;**

**while** **(**i **!=** 1**)** **{**

//-1：如果小于父节点 ，交换并继续判断

**if** **(**tree**.**get**(**i**).**compareTo**(**tree**.**get**(**i**/**2**))** **==** RootType**)** **{**

int temp **=** tree**.**get**(**i**);**

tree**.**set**(**i**,** tree**.**get**(**i**/**2**));**

tree**.**set**(**i**/**2**,** temp**);**

i **=** i**/**2**;**

**}** **else** **{**//小于父节点，退出判断

**break;**

**}**

**}**

**}**

static int deleteTop**(**List**<**Integer**>** tr**)** **{**

int top **=** tr**.**get**(**1**);**

tr**.**set**(**1**,** tr**.**get**(**tr**.**size**()-**1**));**

tr**.**remove**(**tr**.**size**()-**1**);**

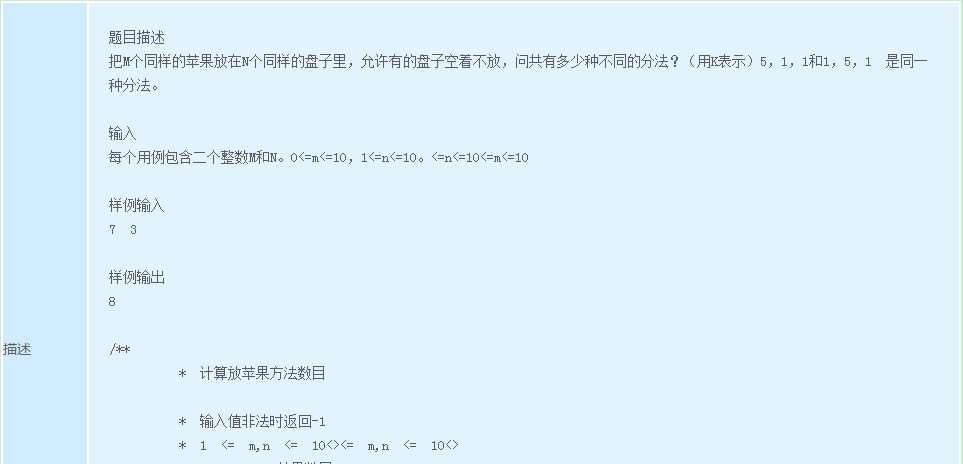
siftDown**(**1**,** tr**);**

**return** top**;**

**}**

**}**

# 放苹果



package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Collections**;**

**import** java**.**util**.**HashSet**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

static int dishTotal **=** 0**,**appleTotal**=**0**;**

static int**[]** appleInDish**;**

//存放不重复的分配方案

static HashSet**<**ArrayList**<**Integer**>>** methods **=** **new** HashSet**<>();**

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

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

**while** **(**scanner**.**hasNextInt**())** **{**

appleTotal **=** scanner**.**nextInt**();**

dishTotal **=** scanner**.**nextInt**();**

appleInDish **=** **new** int**[**dishTotal**+**1**];**

// dfs(1, 0);//从第一个盘子开始分配，此时所有盘子上总的苹果个数为0

// System.out.println(methods.size());

// methods.clear();

System**.**out**.**println**(**getMN**(**appleTotal**,** dishTotal**));**

**}**

scanner**.**close**();**

**}**

//网上大神的算法：递归实现

static int getMN**(**int m**,** int n**)**

**{**

**if(**m **==** 0 **||** n **==** 1**)**

**return** 1**;**

// 苹果数少于盘子数

**if(**m **<** n**)**

**return** getMN**(**m**,** m**);**

**else**

// 有空盘 + 无空盘

**return** getMN**(**m**,** n **-** 1**)** **+** getMN**(**m **-** n**,** n**);**

**}**

//使用dfs算法对每个盘子上的苹果数目分配

//int dishNum:盘子编号 1~N，int apple：当前分配过的盘子上的总苹果个数

//算法复杂度M\*N

static void dfs**(**int dishNum**,** int apple**)** **{**

//对最后一个盘子分配苹果

**if** **(**dishNum **==** dishTotal**)** **{**

appleInDish**[**dishNum**]** **=** appleTotal **-** apple**;**

ArrayList**<**Integer**>** ap **=** **new** ArrayList**<>();**

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

ap**.**add**(**appleInDish**[**i**]);**

**}**

//将该分配方案排序后放入set，即可实现无重复添加

Collections**.**sort**(**ap**);**

methods**.**add**(**ap**);**

// System.out.println(ap);

appleInDish**[**dishNum**]** **=** 0**;**

**return;**

**}**

//对1~N-1编号盘子分配苹果，因为类似 5 1 1，1 1 5这2种视为同一种方式

//故将第一个盘子的最少苹果数目控制在

//总的苹果数量的一半(好像有时候不对，所以改为0)

**for** **(**int i**=**appleTotal**;** i**>=**0**;** i**--)** **{**

**if** **(**apple **+** i **<=** appleTotal**)** **{**

appleInDish**[**dishNum**]** **=** i**;**

dfs**(**dishNum**+**1**,** apple**+**i**);**

appleInDish**[**dishNum**]** **=** 0**;**

**}**

**}**

**}**

**}**

# 24点游戏算法



**import** java**.**util**.**Scanner**;**

public class Main **{**

static boolean endFlag **=** **false;**

static int times **=** 0**;**

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

int a**[]** **=** **new** int**[**4**];**

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

**while** **(**scanner**.**hasNext**())** **{**

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

a**[**i**]** **=** scanner**.**nextInt**();**

**}**

dfs**(**a**,** 0**,** 0**);**

times **=** 0**;**

endFlag **=** **false;**

**}**

scanner**.**close**();**

**}**

static void dfs**(**int**[]** nums**,**int index**,** int res**)** **{**

**if** **(**endFlag**)** **return;**

**if** **(**index **==** nums**.**length**)** **{**

// System.out.println(res);

**if** **(**res **==** 24**)** **{**

System**.**out**.**print**(true);**

endFlag **=** **true;**

**}**

times**++;**

**if** **(**times **==** 64**)**

System**.**out**.**print**(false);**

**return;**

**}**

int resTemp**;**

**if** **(**index **==** 0**)** **{**

resTemp **=** nums**[**index**]** **+** nums**[**index**+**1**];**

dfs**(**nums**,** index**+**2**,** resTemp**);**

resTemp **=** nums**[**index**]** **-** nums**[**index**+**1**];**

dfs**(**nums**,** index**+**2**,** resTemp**);**

resTemp **=** nums**[**index**]** **\*** nums**[**index**+**1**];**

dfs**(**nums**,** index**+**2**,** resTemp**);**

resTemp **=** nums**[**index**]** **/** nums**[**index**+**1**];**

dfs**(**nums**,** index**+**2**,** resTemp**);**

**}** **else** **{**

resTemp **=** res **+** nums**[**index**];**

dfs**(**nums**,** index**+**1**,** resTemp**);**

resTemp **=** res **-** nums**[**index**];**

dfs**(**nums**,** index**+**1**,** resTemp**);**

resTemp **=** res **\*** nums**[**index**];**

dfs**(**nums**,** index**+**1**,** resTemp**);**

resTemp **=** res **/** nums**[**index**];**

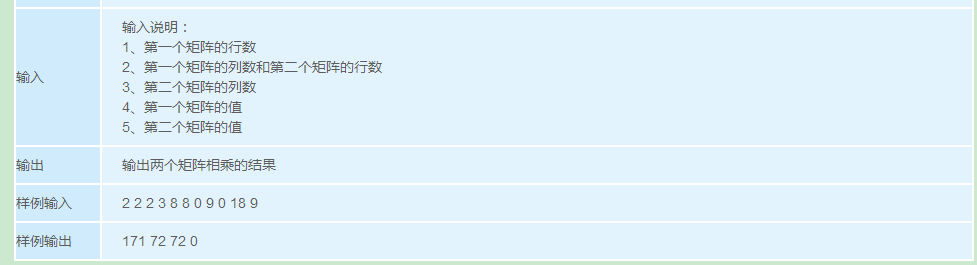
dfs**(**nums**,** index**+**1**,** resTemp**);**

**}**

**}**

**}**

# 矩阵乘法



**import** java**.**util**.**Scanner**;**

public class Main **{**

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

int**[][]** a **=** **{{**3**,**8**},{**8**,**0**}};**

int**[][]** b **=** **{{**9**,**0**},{**18**,**9**}};**

int x**=**a**.**length**,** y**=**a**[**0**].**length**,** z**=**2**;**

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

**while** **(**scanner**.**hasNextInt**())** **{**

x **=** scanner**.**nextInt**();**

y **=** scanner**.**nextInt**();**

z **=** scanner**.**nextInt**();**

a **=** **new** int**[**x**][**y**];**

b **=** **new** int**[**y**][**z**];**

int**[][]** c**=** **new** int**[**x**][**z**];**

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

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

a**[**j**][**i**]** **=** scanner**.**nextInt**();**

**}**

**}**

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

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

b**[**j**][**i**]** **=** scanner**.**nextInt**();**

**}**

**}**

MatrixMultiple**(**a**,** b**,** c**);**

**}**

scanner**.**close**();**

**}**

private static void MatrixMultiple**(**int**[][]** a**,** int**[][]** b**,** int**[][]** c**)** **{**

int x **=** a**.**length**,** y **=** a**[**0**].**length**,** z **=** c**[**0**].**length**;**

**for** **(**int k**=**0**;** k**<**z**;** k**++)** **{**

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

// int sum = 0;

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

c**[**j**][**k**]** **+=** a**[**j**][**i**]\***b**[**i**][**k**];**

**}**

// c[j][k] = sum;

// System.out.println(sum);

**}**

**}**

printMatrix**(**c**);**

**}**

static void printMatrix**(**int**[][]** m**)** **{**

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

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

**if** **(**i **==** m**[**0**].**length **-** 1**)**

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

**else**

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

**}**

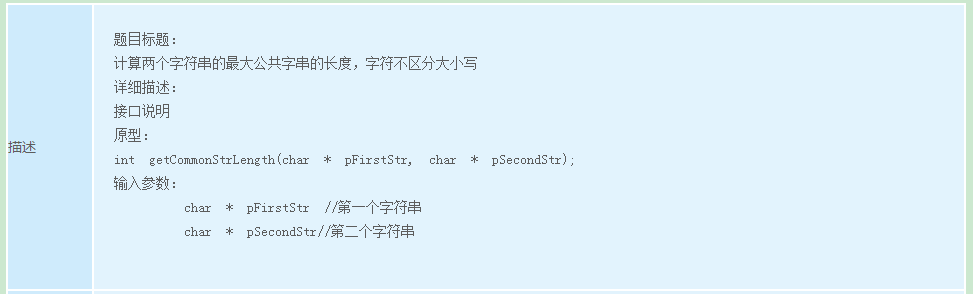
System**.**out**.**println**();**

**}**

**}**

**}**

# 公共字串计算



package com**.**oj**;**

注意最长公共子串（Longest CommonSubstring）和最长公共子序列（LongestCommon Subsequence, LCS）的区别：子串（Substring）是串的一个连续的部分，子序列（Subsequence）则是从不改变序列的顺序，而从序列中去掉任意的元素而获得的新序列；更简略地说，前者（子串）的字符的位置必须连续，后者（子序列LCS）则不必。比如字符串acdfg同akdfc的最长公共子串为df，而他们的最长公共子序列是adf。LCS可以使用动态规划法解决。

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**sc**.**hasNext**()){**

String str1**=**sc**.**next**().**toLowerCase**();**

String str2**=**sc**.**next**().**toLowerCase**();**

System**.**out**.**println**(**lcs2Dp2**(**str1**,** str2**,**str1**.**length**(),** str2**.**length**()));**

System**.**out**.**println**(**lcs2Dp**(**str1**,** str2**));**

**}**

**}**

//求最长公共子串

private static String lcsDp**(**String str1**,** String str2**)** **{**

**if(**str1**.**length**()>**str2**.**length**()){**

String str**=**str1**;**

str1**=**str2**;**

str2**=**str**;**

**}**

int**[][]** dp**=new** int**[**str1**.**length**()+**1**][**str2**.**length**()+**1**];**

int maxLen**=**0**;**

int maxIndex**=**0**;**

**for(**int i**=**1**;** i**<=**str1**.**length**();** i**++){**

**for** **(**int j**=**1**;** j**<=**str2**.**length**();** j**++){**

**if(**str1**.**charAt**(**i**-**1**)** **==** str2**.**charAt**(**j**-**1**)){**

dp**[**i**][**j**]** **=** dp**[**i**-**1**][**j**-**1**]+**1**;**

**}**

**if(**dp**[**i**][**j**]** **>** maxLen**){**

maxLen **=** dp**[**i**][**j**];**

maxIndex **=** i**;**

**}**

**}**

**}**

**return** **(**str1**.**substring**(**maxIndex**-**maxLen**,** maxIndex**));**

**}**

//求最长公共子序列长度,len1是s1字符串的长度

static int lcs2Dp2**(**String s1**,** String s2**,**int len1**,** int len2**)** **{**

**if** **(**len1 **==** 0 **||** len2 **==** 0**)** **return** 0**;**

**if** **(**s1**.**charAt**(**len1**-**1**)** **==** s2**.**charAt**(**len2**-**1**))** **{**

**return** lcs2Dp2**(**s1**,** s2**,** len1**-**1**,** len2**-**1**)** **+** 1**;**

**}** **else** **{**

**return** max**(**lcs2Dp2**(**s1**,** s2**,** len1**,** len2**-**1**),**lcs2Dp2**(**s1**,** s2**,** len1**-**1**,** len2**));**

**}**

**}**

private static int max**(**int n1**,** int n2**)** **{**

**if** **(**n1 **>** n2**)**

**return** n1**;**

**else**

**return** n2**;**

**}**

//求最长公共子序列

static String lcs2Dp**(**String s1**,** String s2**)** **{**

int len1 **=** s1**.**length**(),** len2 **=** s2**.**length**();**

int dp**[][]** **=** **new** int**[**len1**+**1**][**len2**+**1**];**

**for** **(**int i**=**1**;** i**<=**len1**;** i**++)** **{**

**for** **(**int j**=**1**;** j**<=**len2**;** j**++)** **{**

**if** **(**s1**.**charAt**(**i**-**1**)** **==** s2**.**charAt**(**j**-**1**))** **{**

dp**[**i**][**j**]** **=** dp**[**i**-**1**][**j**-**1**]** **+** 1**;**

**}** **else** **if** **(**dp**[**i**][**j**-**1**]** **>** dp**[**i**-**1**][**j**])** **{**

dp**[**i**][**j**]** **=** dp**[**i**][**j**-**1**];**

**}** **else** **{**

dp**[**i**][**j**]** **=** dp**[**i**-**1**][**j**];**

**}**

**}**

**}**

//最长长度值在dp[len1][len2]中

**return** getLcsStr**(**dp**,** s1**,** s2**);**

**}**

static String getLcsStr**(**int**[][]** dp**,** String s1**,** String s2**)** **{**

int len1 **=** s1**.**length**(),** len2 **=** s2**.**length**();**

StringBuilder sb **=** **new** StringBuilder**();**

**while** **(**len1 **>** 0 **&&** len2 **>** 0**)** **{**

**if** **(**s1**.**charAt**(**len1**-**1**)** **==** s2**.**charAt**(**len2**-**1**))** **{**

sb**.**insert**(**0**,** s1**.**charAt**(**len1**-**1**));**

len1**--;**len2**--;**

**}** **else** **{**

**if** **(**dp**[**len1**][**len2**-**1**]** **==** dp**[**len1**][**len2**])** **{**

len2**--;**

**}** **else** **{**

len1**--;**

**}**

**}**

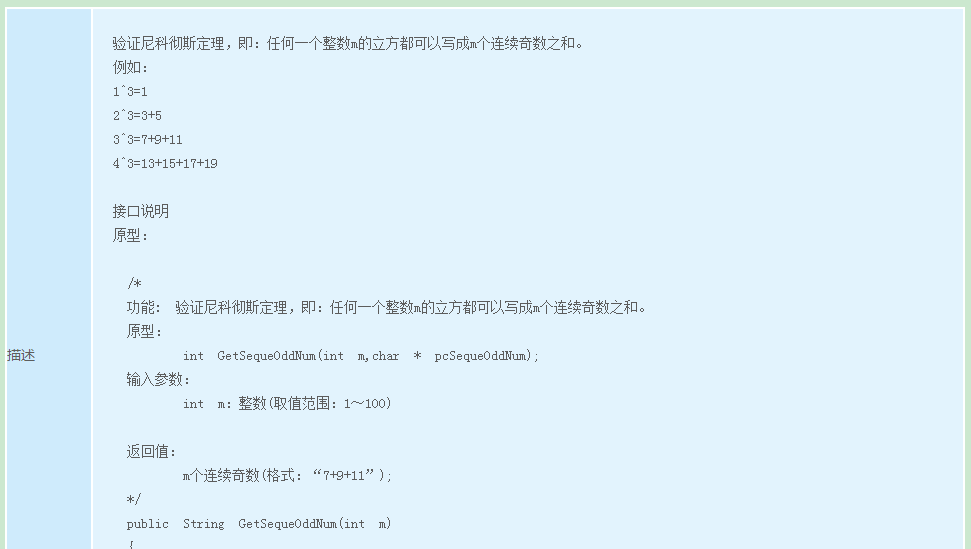
**}**

**return** **(**sb**.**toString**());**

**}**

**}**

# 尼科彻斯定理



**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**sc**.**hasNext**()){**

int num **=** sc**.**nextInt**();**

int pow3 **=** getPower3**(**num**);**

int start **=** pow3**/**num **-** num**;**

**for** **(**int i**=**start**;** i**<**pow3**;** i**++)** **{**

**if** **(**i **%** 2 **==** 1**)** **{**

**if** **(**getSum**(**i**,** num**)** **==** pow3**)** **{**

print**(**i**,** num**);**

**break;**

**}**

**}**

**}**

**}**

**}**

//获得num的3次方

static int getPower3**(**int num**)** **{**

**return** num**\***num**\***num**;**

**}**

//获取num之后的n个奇数的和,包括num

static int getSum**(**int num**,** int n**)** **{**

int sum **=** 0**;**

int i **=** 1**;**

**while** **(**i**++** **<=** n**)** **{**

sum **+=** num**;**

num **+=** 2**;**

**}**

**return** sum**;**

**}**

//打印num之后的n个奇数，包括num

static void print**(**int num**,** int n**)** **{**

int i **=** 1**;**

**while** **(**i**++** **<** n**)** **{**

System**.**out**.**print**(**num**+**"+"**);**

num **+=** 2**;**

**}**

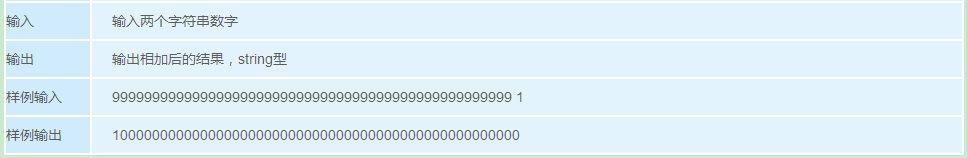
System**.**out**.**print**(**num**);**

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

**}**

**}**

# 超长正整数相加



**import** java**.**math**.**BigInteger**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**sc**.**hasNext**()){**

String add1 **=** sc**.**next**();**

String add2 **=** sc**.**next**();**

System**.**out**.**println**(**bigNumAdd**(**add1**,** add2**));**

//方法2

// BigInteger b1 = new BigInteger(add1);

// BigInteger b2 = new BigInteger(add2);

// b1 = b1.add(b2);

// System.out.println(b1);

**}**

**}**

private static String bigNumAdd**(**String add1**,** String add2**)** **{**

int lastId1 **=** add1**.**length**()-**1**,** lastId2 **=** add2**.**length**()-**1**;**

StringBuilder sb **=** **new** StringBuilder**();**

int n1**,**n2**,**res**,**cin**=**0**;**

//对2个整数相加，直到其中一个整数加完

**while** **(**lastId1 **>=** 0 **&&** lastId2 **>=** 0**)** **{**

n1 **=** Integer**.**valueOf**(**add1**.**substring**(**lastId1**,** lastId1**+**1**));**

n2 **=** Integer**.**valueOf**(**add2**.**substring**(**lastId2**,** lastId2**+**1**));**

lastId1**--;**lastId2**--;**

res **=** **(**n1**+**n2**+**cin**)** **%** 10**;**

cin **=** **(**n1**+**n2**+**cin**)** **/** 10**;**

sb**.**insert**(**0**,** String**.**valueOf**(**res**));**

**}**

//将较长的整数剩余部分添加到和

**while** **(**lastId1 **>=** 0**)** **{**

n1 **=** Integer**.**valueOf**(**add1**.**substring**(**lastId1**,** lastId1**+**1**));**

lastId1**--;**

res **=** **(**n1**+**cin**)** **%** 10**;**

cin **=** **(**n1**+**cin**)** **/** 10**;**

sb**.**insert**(**0**,** String**.**valueOf**(**res**));**

**}**

**while** **(**lastId2 **>=** 0**)** **{**

n2 **=** Integer**.**valueOf**(**add2**.**substring**(**lastId2**,** lastId2**+**1**));**

lastId2**--;**

res **=** **(**n2**+**cin**)** **%** 10**;**

cin **=** **(**n2**+**cin**)** **/** 10**;**

sb**.**insert**(**0**,** String**.**valueOf**(**res**));**

**}**

//如果2个整数等长，最后有进位，添加

**if** **(**cin **!=** 0**)** **{**

sb**.**insert**(**0**,** String**.**valueOf**(**cin**));**

**}**

**return** **(**sb**.**toString**());**

**}**

**}**

# 计算字符串的距离



package com**.**oj**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**sc**.**hasNext**()){**

String s1 **=** sc**.**next**();**

String s2 **=** sc**.**next**();**

int len **=** **(**s1**.**length**()** **>** s2**.**length**())** **?** s1**.**length**()** **:** s2**.**length**();**

System**.**out**.**println**(**distance2**(**s1**,**0**,** s2**,**0**));**

System**.**out**.**println**(**distance**(**s1**,** s2**));**

**}**

**}**

static int distance**(**String s1**,** String s2**)** **{**

int len1 **=** s1**.**length**(),** len2 **=** s2**.**length**();**

int**[][]** dp **=** **new** int**[**len1**+**1**][**len2**+**1**];**

**for** **(**int i**=**0**;** i**<=**s1**.**length**();** i**++)** **{**

dp**[**i**][**0**]** **=** i**;**

**}**

**for** **(**int i**=**0**;** i**<=**s2**.**length**();** i**++)** **{**

dp**[**0**][**i**]** **=** i**;**

**}**

**for** **(**int i**=**1**;** i**<=**len1**;** i**++)** **{**

**for** **(**int j**=**1**;** j**<=**len2**;** j**++)** **{**

**if** **(**s1**.**charAt**(**i**-**1**)** **==** s2**.**charAt**(**j**-**1**))** **{**

dp**[**i**][**j**]** **=** dp**[**i**-**1**][**j**-**1**];**

**}** **else** **{**

dp**[**i**][**j**]** **=** min**(**dp**[**i**-**1**][**j**-**1**],**dp**[**i**-**1**][**j**],**dp**[**i**][**j**-**1**])+**1**;**

**}**

**}**

**}**

**return** dp**[**len1**][**len2**];**

**}**

// 递归程序。

// 不需要转化的：

// f(i,j)=f(i+1,j+1);

// 在需要转化的步骤中：

// f(i,j)=min(f(i+1,j+1)+1,f(i+1,j)+1,f(i,j+1)+1);

static int distance2**(**String a**,**int aStart**,** String b**,**int bStart**)** **{**

int aEnd **=** a**.**length**(),** bEnd **=** b**.**length**();**

**if** **(**aStart **==** aEnd**)** **{**

**if** **(**bStart **==** bEnd**)** **{**

**return** 0**;**

**}** **else** **{**

**return** bEnd**-**bStart**;**

**}**

**}**

**if** **(**bStart **==** bEnd**)** **{**

**if** **(**aStart **==** aEnd**)** **{**

**return** 0**;**

**}** **else** **{**

**return** aEnd**-**aStart**;**

**}**

**}**

**if** **(**a**.**charAt**(**aStart**)** **==** b**.**charAt**(**bStart**))** **{**

**return** distance2**(**a**,** aStart**+**1**,** b**,** bStart**+**1**);**

**}** **else** **{**

**return** min**(**distance2**(**a**,** aStart**+**1**,** b**,** bStart**)**

**,**distance2**(**a**,** aStart**+**1**,** b**,** bStart**+**1**)**

**,**distance2**(**a**,** aStart**+**1**,** b**,** bStart**+**1**))+**1**;**

**}**

**}**

private static int min**(**int i**,** int j**,** int k**)** **{**

int m **=** i **<** j **?** i **:** j**;**

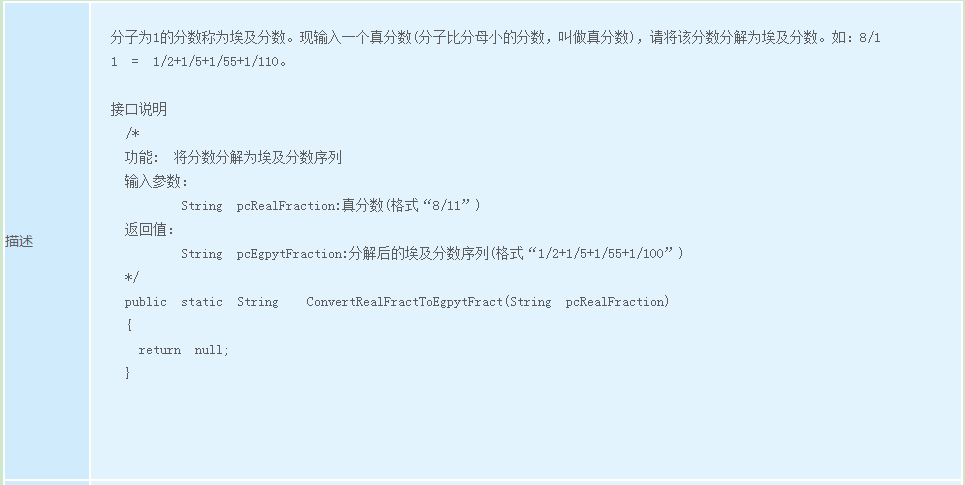
m **=** m **<** k **?** m **:** k**;**

**return** m**;**

**}**

**}**

# 将真分数分解为埃及分数



**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String numStr **=** scanner**.**next**();**

String**[]** numStrs **=** numStr**.**split**(**"/"**);**

long up **=** Integer**.**valueOf**(**numStrs**[**0**]);**

long down **=** Integer**.**valueOf**(**numStrs**[**1**]);**

System**.**out**.**println**(**getResult2**(**up**,** down**));**

**}**

scanner**.**close**();**

**}**

static String getResult**(**long up**,**long down**)** **{**

StringBuilder sb **=** **new** StringBuilder**();**

long temp **=** 0**;**

**while** **(**down **%** up **!=** 0**)** **{**

//得到第一个埃及分数的分母

temp **=** down**/**up **+** 1**;**

sb**.**append**(**1**+**"/"**+**temp**+**"+"**);**

//公式：up/down - 1/temp

up **=** up**\***temp **-** down**;**

down **=** down**\***temp**;**

**}**

sb**.**append**(**1**+**"/"**+**down**/**up**);**

**return** sb**.**toString**();**

**}**

static String getResult2**(**long up**,**long down**)** **{**

StringBuilder sb **=** **new** StringBuilder**();**

long temp **=** 0**;**

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

**if** **(**up **==** 3 **&&** down **%** 2 **==** 0**)** **{**

sb**.**append**(**1**+**"/"**+**down**/**2**+**"+"**+**1**+**"/"**+**down**);**

**break;**

**}**

**if** **(**down **%** up **==** 0**)** **{**

sb**.**append**(**1**+**"/"**+**down**/**up**);**

**break;**

**}** **else** **{**

temp **=** down**/**up **+** 1**;**

sb**.**append**(**1**+**"/"**+**temp**+**"+"**);**

**}**

//公式：up/down - 1/temp

up **=** up**\***temp **-** down**;**

down **=** down**\***temp**;**

**}**

**return** sb**.**toString**();**

**}**

**}**

# 判断IP合法

**import** java**.**util**.**Scanner**;**

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

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

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String ip **=** scanner**.**next**();**

Pattern pattern **=** Pattern**.**compile**(**"(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"

**+** "\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"

**+** "\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"

**+** "\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"**);**

Matcher matcher **=** pattern**.**matcher**(**ip**);**

System**.**out**.**println**(**matcher**.**matches**());**

**}**

scanner**.**close**();**

**}**

**}**

# 数组分2组



**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**HashSet**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

static int findSum **=** 88**;**

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

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

**while** **(**scanner**.**hasNext**())** **{**

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

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

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

all**[**i**]** **=** scanner**.**nextInt**();**

**}**

int sumFive **=** 0**,**sumThree **=** 0**,**sumOther**=**0**,**differece**=**0**;**

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

int index**=**0**;**

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

**if** **(**all**[**i**]** **%** 5 **==** 0**)** **{**

sumFive **+=** all**[**i**];**//对所有5的倍数求和

**}** **else** **if** **(**all**[**i**]** **%** 3 **==** 0**)** **{**

sumThree **+=** all**[**i**];**//对所有3的倍数求和

**}** **else** **{**//对剩余其他数放到一组，并求和

other**[**index**++]** **=** all**[**i**];**

sumOther **+=** all**[**i**];**

**}**

**}**

//5的倍数的所有数之和 与 3的倍数的所有数值和的差值

differece **=** Math**.**abs**(**sumFive **-** sumThree**);**

//将剩余的数分成2组，m1,m2分别表示这2组数的和

//应满足如下条件：假设m1>m2

//m1+m2 = sumOther, m1-m2 = differece

//故m1 = (sumOther+differece)/2; 且m1为整数

//找出sumOther数组中能否分出一组数，使其和为m1

**if** **((**sumOther**+**differece**)** **%** 2 **==** 1**)** **{**

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

**continue;**

**}**

int m1 **=** **(**sumOther**+**differece**)/**2**;**

findSum **=** m1**;**

findFlag **=** **false;**//连续判断前先复位

getPartSum**(**other**,** 0**,** 0**);**

**if** **(**findFlag**)**

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

**else**

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

**}**

scanner**.**close**();**

**}**

// static HashSet<Integer> sumSet = new HashSet<>();

volatile static boolean findFlag **=** **false;**

//找出sumOther数组中能否分出一组数，使其和为m1

static void getPartSum**(**int**[]** arr**,** int index**,** int sum**)** **{**

**if** **(**findFlag**)** **return;**

// sumSet.add(sum);

**if** **(**sum **==** findSum**)** **{**

findFlag **=** **true;**

**}**

**if** **(**index **==** arr**.**length**)** **{**

**return;**

**}**

int tempSum **=** 0**;**

//加上该元素

tempSum **=** sum **+** arr**[**index**];**

getPartSum**(**arr**,** index**+**1**,** tempSum**);**

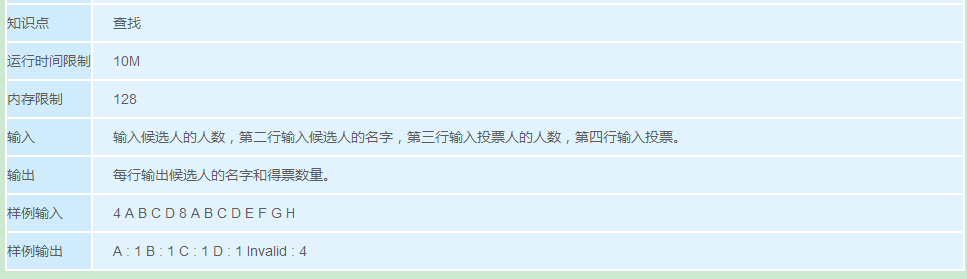
//不加该元素

getPartSum**(**arr**,** index**+**1**,** sum**);**

**}**

**}**

# 记票统计



package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

int candidateNum **=** scanner**.**nextInt**();**

ArrayList**<**String**>** candidates **=** **new** ArrayList**<>();**

int**[]** vote **=** **new** int**[**candidateNum**];**

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

candidates**.**add**(**scanner**.**next**());**

**}**

int voteNum **=** scanner**.**nextInt**();**

int validVotes**=**0**;**

String voteName**;**

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

voteName **=** scanner**.**next**();**

**if** **(**candidates**.**contains**(**voteName**))** **{**

int id **=** candidates**.**indexOf**(**voteName**);**

vote**[**id**]++;**

validVotes**++;**

**}**

**}**

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

**{**

System**.**out**.**println**(**candidates**.**get**(**i**)+**" : "**+**vote**[**i**]);**

**}**

System**.**out**.**println**(**"Invalid : "**+(**voteNum**-**validVotes**));**

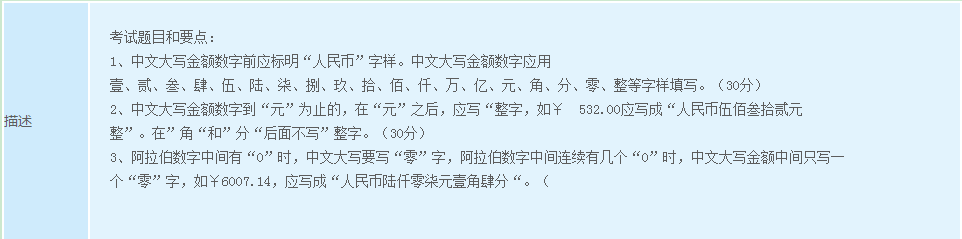
**}**

scanner**.**close**();**

**}**

**}**

# 人民币转换



**import** java**.**util**.**Scanner**;**

public class Main **{**

static String**[]** num **=** **{**"零"**,**"壹"**,**"贰"**,**"叁"**,**"肆"**,**"伍"**,**"陆"**,**"柒"**,**"捌"**,**"玖"**,**"拾"**};**

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

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

**while** **(**scanner**.**hasNext**())** **{**

double money **=** scanner**.**nextFloat**();**

System**.**out**.**println**(**"人民币"**+**getRMB**((**long**)** money**)+**getRMBDecimal**(**money**));**

**}**

scanner**.**close**();**

**}**

static String getRMB**(**long m**)** **{**

//拾、佰、仟、万、亿、元、角、分、零、整

**if** **(**m**>=**0 **&&** m **<=** 9**)** **{**

**return** **(**num**[(**int**)** m**]);**

**}** **else** **if** **(**m **<** 100**)** **{**

**if** **(**m **%** 10 **==** 0**)** **{**

**return** **(**m**/**10 **==** 1**)?**"拾"**:**getRMB**(**m**/**10**)** **+** "拾"**;**

**}** **else** **{**

**return** **(**m**/**10 **==** 1**)?**"拾"

**+** getRMB**(**m**%**10**):**getRMB**(**m**/**10**)** **+** "拾" **+** getRMB**(**m**%**10**);**

**}**

**}** **else** **if** **(**m **<** 1000**)** **{**

**if** **(**m **%** 100 **==** 0**)** **{**

**return** getRMB**(**m**/**100**)** **+** "佰"**;**

**}else** **if** **(**m **%** 100 **<** 10**)** **{** //十位为0

**return** getRMB**(**m**/**100**)** **+** "佰零"**+**getRMB**(**m**%**10**);**

**}else** **{**

**return** getRMB**(**m**/**100**)** **+** "佰" **+** getRMB**(**m**%**100**);**

**}**

**}** **else** **if** **(**m **<** 10000**)** **{**

**if** **(**m **%** 1000 **==** 0**)** **{**

**return** getRMB**(**m**/**1000**)** **+** "仟"**;**

**}** **else** **if** **(**m **%** 1000 **<** 100**)** **{**//百位为0

**return** getRMB**(**m**/**1000**)** **+** "仟零"**+**getRMB**(**m**%**100**);**

**}** **else** **{**

**return** getRMB**(**m**/**1000**)** **+** "仟" **+** getRMB**(**m**%**1000**);**

**}**

**}** **else** **if** **(**m **<** 100\_000\_000L**)** **{**//小于1亿

**if** **(**m **%** 10000 **==** 0**)** **{**

**return** getRMB**(**m**/**10000**)** **+** "万"**;**

**}** **else** **{**

**return** getRMB**(**m**/**10000**)** **+** "万" **+** getRMB**(**m**%**10000**);**

**}**

**}** **else** **if** **(**m **>=** 100\_000\_000L**){**

**if** **(**m **%** 100\_000\_000L **==** 0**)** **{**

**return** getRMB**(**m**/**100\_000\_000L**)** **+** "亿"**;**

**}** **else** **{**

**return** getRMB**(**m**/**100\_000\_000L**)** **+** "亿" **+** getRMB**(**m**%**100\_000\_000L**);**

**}**

**}**

**return** ""**;**

**}**

static String getRMBDecimal**(**double money**)** **{**

int decimal **=** **(**int**)** **(**money**\***100 **-** **((**int**)(**money**))\***100**);**

**if** **(**decimal **==** 0**)** **{**

**return** "元整"**;**

**}** **else** **if** **(**decimal **<** 10**){**

**return** "元"**+**num**[**decimal**%**10**]+**"分"**;**

**}** **else** **{**

**return** "元"**+**num**[**decimal**/**10**]+**"角"

**+(((**decimal**%**10**)==**0**)?**""**:**num**[**decimal**%**10**]+**"分"**);**

**}**

**}**

**}**

改进后：

**import** java**.**util**.**Scanner**;**

public class Main **{**

static String**[]** num **=** **{**"零"**,**"壹"**,**"贰"**,**"叁"**,**"肆"**,**"伍"**,**"陆"**,**"柒"**,**"捌"**,**"玖"**,**"拾"**};**

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

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

**while** **(**scanner**.**hasNext**())** **{**

double money **=** scanner**.**nextDouble**();**

System**.**out**.**println**(**"人民币"**+**

**((**getRMB**((**long**)** money**).**equals**(**"零"**))?**""**:**getRMB**((**long**)** money**)+**"元"**)**

**+**getRMBDecimal**(**money**));**

**}**

scanner**.**close**();**

**}**

static String getRMB**(**long m**)** **{**

//拾、佰、仟、万、亿、元、角、分、零、整

**if** **(**m**>=**0 **&&** m **<=** 9**)** **{**

**return** **(**num**[(**int**)** m**]);**

**}** **else** **if** **(**m **<** 100**)** **{**

**if** **(**m **%** 10 **==** 0**)** **{**

**return** **(**m**/**10 **==** 1**)?**"拾"**:**getRMB**(**m**/**10**)** **+** "拾"**;**

**}** **else** **{**

**return** **(**m**/**10 **==** 1**)?**"拾"

**+** getRMB**(**m**%**10**):**getRMB**(**m**/**10**)** **+** "拾" **+** getRMB**(**m**%**10**);**

**}**

**}** **else** **if** **(**m **<** 1000**)** **{**

**if** **(**m **%** 100 **==** 0**)** **{**

**return** getRMB**(**m**/**100**)** **+** "佰"**;**

**}else** **if** **(**m **%** 100 **<** 10**)** **{** //十位为0

**return** getRMB**(**m**/**100**)** **+** "佰零"**+**getRMB**(**m**%**10**);**

**}else** **{**

**return** getRMB**(**m**/**100**)** **+** "佰" **+** getRMB**(**m**%**100**);**

**}**

**}** **else** **if** **(**m **<** 10000**)** **{**

**if** **(**m **%** 1000 **==** 0**)** **{**

**return** getRMB**(**m**/**1000**)** **+** "仟"**;**

**}** **else** **if** **(**m **%** 1000 **<** 100**)** **{**//百位为0

**return** getRMB**(**m**/**1000**)** **+** "仟零"**+**getRMB**(**m**%**100**);**

**}** **else** **{**

**return** getRMB**(**m**/**1000**)** **+** "仟" **+** getRMB**(**m**%**1000**);**

**}**

**}** **else** **if** **(**m **<** 100\_000\_000L**)** **{**//小于1亿

**if** **(**m **%** 10000 **==** 0**)** **{**

**return** getRMB**(**m**/**10000**)** **+** "万"**;**

**}** **else** **{**

**return** getRMB**(**m**/**10000**)** **+** "万" **+** getRMB**(**m**%**10000**);**

**}**

**}** **else** **if** **(**m **>=** 100\_000\_000L**){**

**if** **(**m **%** 100\_000\_000L **==** 0**)** **{**

**return** getRMB**(**m**/**100\_000\_000L**)** **+** "亿"**;**

**}** **else** **{**

**return** getRMB**(**m**/**100\_000\_000L**)** **+** "亿" **+** getRMB**(**m**%**100\_000\_000L**);**

**}**

**}**

**return** ""**;**

**}**

static String getRMBDecimal**(**double money**)** **{**

long temp **=** **(**long**)** **(**money**\***1000**);**

int decimal **=** **(**int**)** **(**temp**/**10**);**

**if** **(**decimal **==** 0**)** **{**

**return** "整"**;**

**}** **else** **if** **(**decimal **<** 10**){**

**return** num**[**decimal**%**10**]+**"分"**;**

**}** **else** **{**

**return** num**[**decimal**/**10**]+**"角"

**+(((**decimal**%**10**)==**0**)?**""**:**num**[**decimal**%**10**]+**"分"**);**

**}**

**}**

**}**

# [动态规划](http://www.hawstein.com/posts/dp-novice-to-advanced.html)

如果我们有面值为1元、3元和5元的硬币若干枚，如何用最少的硬币凑够11元？ (表面上这道题可以用贪心算法，但贪心算法无法保证可以求出解，比如1元换成2元的时候)

首先我们思考一个问题，如何用最少的硬币凑够i元(i<11)？为什么要这么问呢？ 两个原因：1.当我们遇到一个大问题时，总是习惯把问题的规模变小，这样便于分析讨论。 2.这个规模变小后的问题和原来的问题是同质的，除了规模变小，其它的都是一样的， 本质上它还是同一个问题(规模变小后的问题其实是原问题的子问题)。

public class Demo **{**

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

int**[]** dp **=** **new** int**[**12**];**

printArray**(**dp**);**

**for** **(**int i**=**1**;** i**<**12**;** i**++)** **{**

int m1**=**100**,**m2**=**100**,**m3**=**100**;**

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

m1 **=** dp**[**i**-**1**]** **+** 1**;**

**}**

**if** **(**i **>=** 3**)** **{**

m2 **=** dp**[**i**-**3**]** **+** 1**;**

**}**

**if** **(**i **>=** 5**)** **{**

m3 **=** dp**[**i**-**5**]** **+** 1**;**

**}**

dp**[**i**]** **=** min**(**m1**,**m2**,**m3**);**

**}**

printArray**(**dp**);**

**}**

private static int min**(**int m1**,** int m2**,** int m3**)** **{**

int minV **=** m1 **<** m2 **?** m1 **:** m2**;**

minV **=** m3 **<** minV **?** m3 **:** minV**;**

**return** minV**;**

**}**

static void printArray**(**int**[]** ns**)** **{**

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

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

**}**

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

**}**

**}**

**static** **void** get() {

**int**[] dp = **new** **int**[12];

**int** inf = 100;

**for** (**int** i=1; i<=11; i++) {

dp[i] = *min*(i-1>=0?dp[i-1]+1:inf, i-3>=0?dp[i-3]+1:inf,

i-5>=0?dp[i-5]+1:inf);

}

}

**private** **static** **int** min(**int** i, **int** j, **int** k) {

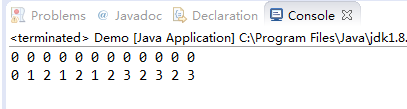
**return** *min2*(i, *min2*(j, k));

}

**private** **static** **int** min2(**int** j, **int** k) {

**return** j < k ? j : k;

}



一个序列有N个数：A[1],A[2],…,A[N]，求出最长非降子序列的长度。 (讲DP基本都会讲到的一个问题LIS：longest increasing subsequence)

让我们沿用“入门”一节里那道简单题的思路来一步步找到“状态”和“状态转移方程”。 假如我们考虑求A[1],A[2],…,A[i]的最长非降子序列的长度，其中i<N， 那么上面的问题变成了原问题的一个子问题(问题规模变小了，你可以让i=1,2,3等来分析) 然后我们定义d(i)，表示前i个数中以A[i]结尾的最长非降子序列的长度。OK， 对照“入门”中的简单题，你应该可以估计到这个d(i)就是我们要找的状态。 如果我们把d(1)到d(N)都计算出来，那么最终我们要找的答案就是这里面最大的那个。 状态找到了，下一步找出状态转移方程。

为了方便理解我们是如何找到状态转移方程的，我先把下面的例子提到前面来讲。 如果我们要求的这N个数的序列是：

5，3，4，8，6，7

根据上面找到的状态，我们可以得到：（下文的最长非降子序列都用LIS表示）

* 前1个数的LIS长度d(1)=1(序列：5)
* 前2个数的LIS长度d(2)=1(序列：3；3前面没有比3小的)
* 前3个数的LIS长度d(3)=2(序列：3，4；4前面有个比它小的3，所以d(3)=d(2)+1)
* 前4个数的LIS长度d(4)=3(序列：3，4，8；8前面比它小的有3个数，所以 d(4)=max{d(1),d(2),d(3)}+1=3)

OK，分析到这，我觉得状态转移方程已经很明显了，如果我们已经求出了d(1)到d(i-1)， 那么d(i)可以用下面的状态转移方程得到：

d(i) = max{1, d(j)+1},其中j<i,A[j]<=A[i]

用大白话解释就是，想要求d(i)，就把i前面的各个子序列中， 最后一个数不大于A[i]的序列长度加1，然后取出最大的长度即为d(i)。 当然了，有可能i前面的各个子序列中最后一个数都大于A[i]，那么d(i)=1， 即它自身成为一个长度为1的子序列。

实际代码中的dp[i]表示数组第i个下标处值到数组起始位置的最长非降序子序列

public class Demo **{**

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

int**[]** ns **=** **{**1**,**3**,**4**,**2**,**5**,**1**,**7**};**

int**[]** dp **=** **new** int**[**ns**.**length**+**1**];**

printArray**(**dp**);**

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

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

dp**[**i**]** **=** 1**;**

**continue;**

**}**

int num **=** ns**[**i**];**

int id **=** i**-**1**;**

int max **=** 0**;**

**while** **(**id **>=** 0**)** **{**

//找到前面小于大于它的所有值中 最长的序列

**if** **(**ns**[**id**]** **<=** num**)** **{**

max **=** **(**dp**[**id**]** **>** max **?** dp**[**id**]** **:** max**);**

**}**

id**--;**

**}**

dp**[**i**]** **=** max**+**1**;**

**}**

printArray**(**dp**);**

**}**

static void printArray**(**int**[]** ns**)** **{**

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

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

**}**

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

**}**

**}**

**static** **void** get(**int**[] array) {

**int**[] dp = **new** **int**[array.length];

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

dp[i] = 1;

}

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

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

**if** (array[k] <= array[i]) {

dp[i] = *max*(dp[i], dp[k]+1);

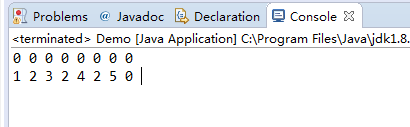
}

}

}

*p*(dp);

}



接下来，让我们来看看如何解决二维的DP问题。

平面上有N\*M个格子，每个格子中放着一定数量的苹果。你从左上角的格子开始， 每一步只能向下走或是向右走，每次走到一个格子上就把格子里的苹果收集起来， 这样下去，你最多能收集到多少个苹果。

解这个问题与解其它的DP问题几乎没有什么两样。第一步找到问题的“状态”， 第二步找到“状态转移方程”，然后基本上问题就解决了。

首先，我们要找到这个问题中的“状态”是什么？我们必须注意到的一点是， 到达一个格子的方式最多只有两种：从左边来的(除了第一列)和从上边来的(除了第一行)。 因此为了求出到达当前格子后最多能收集到多少个苹果， 我们就要先去考察那些能到达当前这个格子的格子，到达它们最多能收集到多少个苹果。 (是不是有点绕，但这句话的本质其实是DP的关键：欲求问题的解，先要去求子问题的解)

经过上面的分析，很容易可以得出问题的状态和状态转移方程。 状态S[i][j]表示我们走到(i, j)这个格子时，最多能收集到多少个苹果。那么， 状态转移方程如下：

S[i][j]=A[i][j] + max(S[i-1][j], if i>0 ; S[i][j-1], if j>0)

其中i代表行，j代表列，下标均从0开始；A[i][j]代表格子(i, j)处的苹果数量。

S[i][j]有两种计算方式：1.对于每一行，从左向右计算，然后从上到下逐行处理；2. 对于每一列，从上到下计算，然后从左向右逐列处理。 这样做的目的是为了在计算S[i][j]时，S[i-1][j]和S[i][j-1]都已经计算出来了。

代码如下：

package com**.**oj**;**

public class Demo **{**

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

int**[][]** dish **=** **{{**1**,**2**,**3**,**4**},{**5**,**6**,**7**,**8**,**9**}};**

int**[][]** dp **=** **new** int**[**3**][**5**];**

int row **=** dish**.**length**;**

int column **=** dish**[**0**].**length**;**

printArray**(**dish**);**

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

**for** **(**int j**=**1**;** j**<=**column**;** j**++)** **{**

dp**[**i**][**j**]** **=** max**(**dp**[**i**-**1**][**j**],** dp**[**i**][**j**-**1**])** **+** dish**[**i**-**1**][**j**-**1**];**

**}**

**}**

printArray**(**dp**);**

System**.**out**.**println**(**"max:"**+**dp**[**row**][**column**]);**

**}**

private static int max**(**int i**,** int j**)** **{**

**return** i **>** j **?** i **:** j**;**

**}**

static void printArray**(**int**[][]** ns**)** **{**

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

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

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

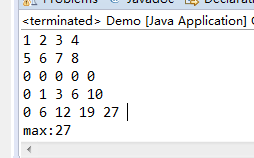
**}**

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

**}**

**}**

**}**



# [背包问题（一）](http://www.hawstein.com/posts/dp-knapsack.html)

话说有一哥们去森林里玩发现了一堆宝石，他数了数，一共有n个。 但他身上能装宝石的就只有一个背包，背包的容量为C。这哥们把n个宝石排成一排并编上号： 0,1,2,…,n-1。第i个宝石对应的体积和价值分别为V[i]和W[i] 。排好后这哥们开始思考： 背包总共也就只能装下体积为C的东西，那我要装下哪些宝石才能让我获得最大的利益呢？

OK，如果是你，你会怎么做？你斩钉截铁的说：动态规划啊！恭喜你，答对了。 那么让我们来看看，动态规划中最最最重要的两个概念： 状态和状态转移方程在这个问题中分别是什么。

我们要怎样去定义状态呢？这个状态总不能是凭空想象或是从天上掉下来的吧。 为了方便说明，让我们先实例化上面的问题。一般遇到n，你就果断地给n赋予一个很小的数， 比如n=3。然后设背包容量C=10，三个宝石的体积为5，4，3，对应的价值为20，10，12。 对于这个例子，我想智商大于0的人都知道正解应该是把体积为5和3的宝石装到背包里， 此时对应的价值是20+12=32。接下来，我们把第三个宝石拿走， 同时背包容量减去第三个宝石的体积（因为它是装入背包的宝石之一）， 于是问题的各参数变为：n=2，C=7，体积｛5，4｝，价值｛20，10｝。好了， 现在这个问题的解是什么？我想智商等于0的也解得出了：把体积为5的宝石放入背包 （然后剩下体积2，装不下第二个宝石，只能眼睁睁看着它溜走），此时价值为20。 这样一来，我们发现，n=3时，放入背包的是0号和2号宝石；当n=2时， 我们放入的是0号宝石。这并不是一个偶然，没错， 这就是传说中的“全局最优解包含局部最优解”（n=2是n=3情况的一个局部子问题）。 绕了那么大的圈子，你可能要问，这都哪跟哪啊？说好的状态呢？说好的状态转移方程呢？ 别急，它们已经呼之欲出了。

我们再把上面的例子理一下。当n=2时，我们要求的是前2个宝石， 装到体积为7的背包里能达到的最大价值；当n=3时，我们要求的是前3个宝石， 装到体积为10的背包里能达到的最大价值。有没有发现它们其实是一个句式！OK， 让我们形式化地表示一下它们， 定义d(i,j)为前i个宝石装到剩余体积为j的背包里能达到的最大价值。 那么上面两句话即为：d(2, 7)和d(3, 10)。这样看着真是爽多了， 而这两个看着很爽的符号就是我们要找的状态了。 即状态d(i,j)表示前i个宝石装到剩余体积为j的背包里能达到的最大价值。 上面那么多的文字，用一句话概括就是：根据子问题定义状态！你找到子问题， 状态也就浮出水面了。而我们最终要求解的最大价值即为d(n, C)：前n个宝石 （0,1,2…,n-1）装入剩余容量为C的背包中的最大价值。状态好不容易找到了， 状态转移方程呢？顾名思义，状态转移方程就是描述状态是怎么转移的方程（好废话！）。 那么回到例子，d(2, 7)和d(3, 10)是怎么转移的？来，我们来说说2号宝石 （记住宝石编号是从0开始的）。从d(2, 7)到d(3, 10)就隔了这个2号宝石。 它有两种情况，装或者不装入背包。如果装入，在面对前2个宝石时， 背包就只剩下体积7来装它们，而相应的要加上2号宝石的价值12， d(3, 10)=d(2, 10-3)+12=d(2, 7)+12；如果不装入，体积仍为10，价值自然不变了， d(3, 10)=d(2, 10)。记住，d(3, 10)表示的是前3个宝石装入到剩余体积为10 的背包里能达到的最大价值，既然是最大价值，就有d(3, 10)=max{ d(2, 10), d(2, 7)+12 }。好了，这条方程描述了状态d(i, j)的一些关系， 没错，它就是状态转移方程了。把它形式化一下：d(i, j)=max{ d(i-1, j), d(i-1,j-V[i-1]) + W[i-1] }。注意讨论前i个宝石装入背包的时候， 其实是在考查第i-1个宝石装不装入背包（因为宝石是从0开始编号的）。至此， 状态和状态转移方程都已经有了。接下来，直接上代码。

好，至此我们解决了背包问题中最基本的0/1背包问题。等等，这时你可能要问， 我现在只知道背包能装入宝石的最大价值，但我还不知道要往背包里装入哪些宝石啊。嗯， 好问题！让我们先定义一个数组x，对于其中的元素为1时表示对应编号的宝石放入背包， 为0则不放入。让我们回到上面的例子，对于体积为5，4，3，价值为20，10，12的3个宝石 ，如何求得其对应的数组x呢？（明显我们目测一下就知道x={1 0 1}， 但程序可目测不出来）OK，让我们还是从状态说起。如果我们把2号宝石放入了背包， 那么是不是也就意味着，前3个宝石放入背包的最大价值要比前2个宝石放入背包的价值大， 即：d(3, 10)>d(2, 10)。再用字母代替具体的数字 （不知不觉中我们就用了不完全归纳法哈），当d(i, j)>d(i-1, j)时，x(i-1)=1;OK， 上代码：

public class Demo **{**

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

int vs**[]** **=** **{**5**,**4**,**3**,**3**};**//宝石体积

int ws**[]** **=** **{**20**,**10**,**12**,**15**};**//宝石重量

int bagVolume **=** 10**;**//背包体积

//状态dp[i][j]表示将前i个宝石装入剩余体积为j的背包中所能获取的最大价值

//状态转移方程为：vs[i]第i个宝石的体积，ws[i]第i个宝石的价值

//dp[i][j]=max{dp[i-1][j](不装入背包),dp[i-1][j-vs[i]]+ws[i](装入)};

int**[][]** dp **=** **new** int**[**vs**.**length**+**1**][**bagVolume**+**1**];**

int row **=** vs**.**length**;**

//背包体积，要求在体积范围内装入价值最多的宝石

int column **=** bagVolume**;**

printArray2**(**dp**);**

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

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

//初始化第一行

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

dp**[**i**][**j**]** **=** 0**;**

**continue;**

**}**

//当前背包剩余体积可以装下第i个宝石。

//vs[i-1]表示第i个宝石的体积，下标从0开始

**if** **(**j **>=** vs**[**i**-**1**])** **{**

dp**[**i**][**j**]** **=** max**(**dp**[**i**-**1**][**j**],** dp**[**i**-**1**][**j**-**vs**[**i**-**1**]]+**ws**[**i**-**1**]);**

**}**

**}**

**}**

//打印列下标

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

System.***out***.printf("%-3d ",i);

}

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

printArray2**(**dp**);**

System**.**out**.**println**(**"max:"**+**dp**[**row**][**column**]);**

//标记装入背包的宝石

int book**[]** **=** **new** int**[**vs**.**length**];**

int j **=** bagVolume**;**

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

**if** **(**dp**[**i**][**j**]** **>** dp**[**i**-**1**][**j**])** **{**

book**[**i**-**1**]** **=** 1**;**

j **=** j **-** vs**[**i**-**1**];**

**}**

**}**

printArray**(**book**);**

**}**

private static int max**(**int i**,** int j**)** **{**

// TODO Auto-generated method stub

**return** i **>** j **?** i **:** j**;**

**}**

static void printArray2**(**int**[][]** ns**)** **{**

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

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

System**.**out**.**printf**(**"%-3d "**,**ns**[**i**][**j**]);**

**}**

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

**}**

**}**

static void printArray**(**int**[]** ns**)** **{**

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

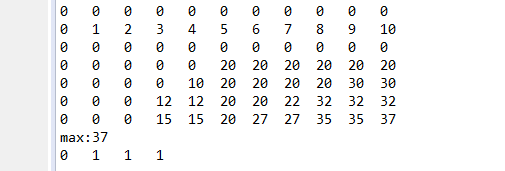
System**.**out**.**printf**(**"%-3d "**,**ns**[**i**]);**

**}**

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

**}**

**}**



由上面那一小段优化过后的代码可知，状态转移方程为：d(i, j)=max{ d(i-1, j), d(i-1, j-V)+W }，也就是在计算d(i, j)时我们用到了d(i-1,j)和d(i-1, j-V)的值。 如果我们只用一个一维数组d(0)～d(C)来保存状态值可以么？将i方向的维数去掉， 我们可以将原来二维数组表示为一维数据：d(i-1, j-V)变为d(j-V)， d(i-1, j)变为d(j)。当我们要计算d(i, j)时，只需要比较d(j)和d(j-V)+W的大小， 用较大的数更新d(j)即可。等等，如果我要计算d(i, j+1)，而它恰好要用到d(i-1, j)的值， 那么问题就出来了，因为你刚刚才把它更新为d(i, j)了。那么，怎么办呢？ 按照j递减的顺序即可避免这种问题。比如，你计算完d(i, j)， 接下来要计算的是d(i,j-1)，而它的状态转移方程为d(i, j-1)=max{ d(i-1, j-1), d(i-1, j-1-V)+W }，它不会再用到d(i-1,j)的值！所以， 即使该位置的值被更新了也无所谓。好，上代码：

public class Demo **{**

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

int vs**[]** **=** **{**5**,**4**,**3**,**3**};**//宝石体积

int ws**[]** **=** **{**20**,**10**,**12**,**15**};**//宝石重量

int bagVolume **=** 10**;**//背包体积

//状态dp[i][j]表示将前i个宝石装入剩余体积为j的背包中所能获取的最大价值

//状态转移方程为：vs[i]第i个宝石的体积，ws[i]第i个宝石的价值

//dp[i][j]=max{dp[i-1][j](不装入背包),dp[i-1][j-vs[i]]+ws[i](装入)};

//为降低空间复杂度 dp[j]=max{dp[j],dp[j-vs[i]]+ws[i]};

//关键：对dp求值应从后往前求

//一维数组的缺点是无法得知哪些宝石需要被装入

int**[]** dp **=** **new** int**[**bagVolume**+**1**];**

int row **=** vs**.**length**;**

//背包体积，要求在体积范围内装入价值最多的宝石

int column **=** bagVolume**;**

printArray**(**dp**);**

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

//关键：对dp求值应从后往前求

**for** **(**int j**=**bagVolume**;** j**>=**0**;** j**--)** **{**

**if** **(**j **>=** vs**[**i**-**1**])** **{**

dp**[**j**]** **=** max**(**dp**[**j**],** dp**[**j**-**vs**[**i**-**1**]]+**ws**[**i**-**1**]);**

**}**

**}**

printArray**(**dp**);**

**}**

//打印列下标

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

System**.**out**.**printf**(**"%-3d "**,**i**);**

**}**

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

printArray**(**dp**);**

System**.**out**.**println**(**"max:"**+**dp**[**bagVolume**]);**

**}**

private static int max**(**int i**,** int j**)** **{**

**return** i **>** j **?** i **:** j**;**

**}**

static void printArray**(**int**[]** ns**)** **{**

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

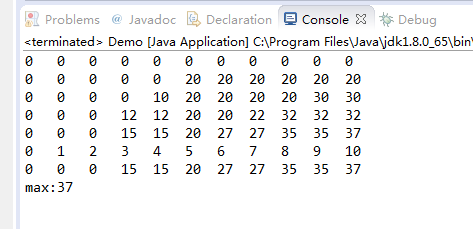
System**.**out**.**printf**(**"%-3d "**,**ns**[**i**]);**

**}**

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

**}**

**}**



# 字符统计



**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Collections**;**

**import** java**.**util**.**HashMap**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNextLine**())** **{**

String in **=** scanner**.**nextLine**();**

int**[]** book **=** **new** int**[**128**];**

char get**;**

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

get **=** in**.**charAt**(**i**);**

**if** **(**String**.**valueOf**(**get**).**matches**(**"[0-9a-zA-z[\\u0020]]"**))**

book**[**in**.**charAt**(**i**)]++;**

**}**

ArrayList**<**CharStatics**>** list **=** **new** ArrayList**<>();**

**for** **(**int i**=**0**;** i**<**128**;** i**++)** **{**

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

CharStatics ch **=** **new** CharStatics**((**char**)** i**,** book**[**i**]);**

list**.**add**(**ch**);**

**}**

**}**

Collections**.**sort**(**list**);**

**for** **(**CharStatics c **:** list**)** **{**

System**.**out**.**print**(**c**.**getName**());**

**}**

**}**

scanner**.**close**();**

**}**

**}**

class CharStatics **implements** Comparable**<**CharStatics**>** **{**

char name**;**

int times**;**

public CharStatics**(**char n**,** int t**)** **{**

name **=** n**;**

times **=** t**;**

**}**

public char getName**()** **{**

**return** name**;**

**}**

@Override

public String toString**()** **{**

**return** name**+**":"**+**times**;**

**}**

@Override

public int compareTo**(**CharStatics o**)** **{**

int d **=** o**.**times **-** **this.**times**;**

**if** **(**d **==** 0**)** **{**

d **=** **this.**name **-** o**.**name**;**

**}**

**return** d**;**

**}**

**}**

# 求解立方根

牛顿迭代法原理：  
对于求a的立方根，可以设f(x)=x^3-a，从而转换成求解f(x)=0，即求方程的根。  
f(x)展开泰勒公式：f(x)=f(x0)+(x-x0)\*f'(x0)=0，得出的x=x0-f(x0)/f'(x0)=g(x0)，此时递归调用该式子可以逐步接近于最终结果。  
为什么会接近于最终结果？当然，牛顿迭代并不是无条件收敛的。

**import** java**.**util**.**Scanner**;**

public class Main **{**

static double PRECISION **=** 0.0001**;**

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

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

**while** **(**scanner**.**hasNext**())** **{**

double n **=** scanner**.**nextDouble**();**

System**.**out**.**printf**(**"%.1f"**,**getCubeRoot**(**n**));**

**}**

scanner**.**close**();**

**}**

static double getCubeRoot**(**double n**)** **{**

double x **=** 1**;**

double x0 **=** x **-** **(**x**\***x**\***x **-** n**)/(**3**\***x**\***x**);**

**while** **(**Math**.**abs**(**x**-**x0**)** **>** PRECISION**)** **{**

x **=** x0**;**

x0 **=** x **-** **(**x**\***x**\***x **-** n**)/(**3**\***x**\***x**);**

**}**

**return** x**;**

**}**

**}**

# 啊哈算法：岛屿染色

public class Main **{**

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

//岛的地图 0代表海水 大于0为陆地

int**[][]** map **=** **{{**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**},**

**{**0**,**1**,**2**,**1**,**0**,**0**,**0**,**0**,**0**,**2**,**3**},**//1~10

**{**0**,**3**,**0**,**2**,**0**,**1**,**2**,**1**,**0**,**1**,**2**},**

**{**0**,**4**,**0**,**1**,**0**,**1**,**2**,**3**,**2**,**0**,**1**},**

**{**0**,**3**,**2**,**0**,**0**,**0**,**1**,**2**,**4**,**0**,**0**},**

**{**0**,**0**,**0**,**0**,**0**,**0**,**0**,**1**,**5**,**3**,**0**},**

**{**0**,**0**,**1**,**2**,**1**,**0**,**1**,**5**,**4**,**3**,**0**},**

**{**0**,**0**,**1**,**2**,**3**,**1**,**3**,**6**,**2**,**1**,**0**},**

**{**0**,**0**,**0**,**3**,**4**,**8**,**9**,**7**,**5**,**0**,**0**},**

**{**0**,**0**,**0**,**0**,**3**,**7**,**8**,**6**,**0**,**1**,**2**},**

**{**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**,**1**,**0**}};**//

int Xborder **=** map**.**length**-**1**;**

int Yborder **=** map**[**0**].**length**-**1**;**

int**[][]** book **=** **new** int**[**Xborder**+**1**][**Yborder**+**1**];**

//使用bfs进行岛屿染色

int color **=** 1**;**

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

**for** **(**int j**=**1**;** j**<=**Yborder**;** j**++)** **{**

**if** **(**map**[**i**][**j**]** **!=** 0 **&&** book**[**i**][**j**]** **==** 0**)** **{**

findIsland**(**map**,** book**,** i**,** j**,** color**++);**

**}**

**}**

**}**

**}**

private static void findIsland**(**int**[][]** map**,**int**[][]** book**,**

int startX**,**int startY**,** int color**)** **{**

MyQueue queue **=** **new** MyQueue**();**

int Xborder **=** map**.**length**-**1**;**

int Yborder **=** map**[**0**].**length**-**1**;**

int sum **=** 0**;**

**if** **(**map**[**startX**][**startY**]** **!=** 0 **&&** book**[**startX**][**startY**]** **!=** 1**)** **{**

queue**.**add**(**startX**,** startY**);**

book**[**startX**][**startY**]** **=** color**;**

sum **=** 1**;**

**}**

**while** **(!**queue**.**isEmputy**())** **{**

int**[][]** next **=** **{{**0**,**1**},{**0**,-**1**},{**1**,**0**},{-**1**,**0**}};**

int**[]** node **=** queue**.**get**();**

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

int toX **=** node**[**0**]** **+** next**[**i**][**0**];**

int toY **=** node**[**1**]** **+** next**[**i**][**1**];**

**if** **(**toX **<** 1 **||** toX **>** Xborder **||** toY **<**1 **||** toY **>** Yborder**)** **{**

**continue;**

**}**

**if** **(**map**[**toX**][**toY**]** **!=** 0 **&&** book**[**toX**][**toY**]** **==** 0**)** **{**

queue**.**add**(**toX**,** toY**);**

book**[**toX**][**toY**]** **=** color**;**

sum**++;**

**}**

**}**

**}**

**if** **(**sum **!=** 0**)** **{**

System**.**out**.**println**(**"color:"**+**color**+**" sum:"**+**sum**);**

printArray2**(**book**);**

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

**}**

**}**

static void printArray2**(**int**[][]** a**)** **{**

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

**for** **(**int j**=**1**;** j**<**a**[**0**].**length**;** j**++)** **{**

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

**}**

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

**}**

**}**

**}**

class MyQueue **{**

private static int MAXSIZE **=** 20**;**

private int**[][]** data**;**

private int head**;**

private int tail**;**

private int size**;**

public MyQueue**()** **{**

data **=** **new** int**[**MAXSIZE**][**2**];**

head **=** 0**;**

tail **=** 0**;**

**}**

public boolean isFull**()** **{**

**if** **((**tail**+**1**)** **%** MAXSIZE **==** head**)**

**return** **true;**

**else**

**return** **false;**

**}**

public boolean isEmputy**()** **{**

**if** **(**tail **==** head**)**

**return** **true;**

**else**

**return** **false;**

**}**

public int getSize**()** **{**

size **=** **(**tail **-** head **+** MAXSIZE**)** **%** MAXSIZE**;**

**return** size**;**

**}**

public boolean add**(**int x**,** int y**)** **{**

**if** **(**isFull**())** **{**

**return** **false;**

**}** **else** **{**

data**[**tail**][**0**]** **=** x**;**

data**[**tail**][**1**]** **=** y**;**

tail **=** **(**tail**+**1**)** **%** MAXSIZE**;**

**return** **true;**

**}**

**}**

public int**[]** get**()** **{**

**if** **(**isEmputy**())** **{**

**return** **null;**

**}** **else** **{**

int**[]** e **=** data**[**head**];**

head **=** **(**head**+**1**)** **%** MAXSIZE**;**

**return** e**;**

**}**

**}**

@Override

public String toString**()** **{**

StringBuilder sb **=** **new** StringBuilder**();**

sb**.**append**(**"h:"**+**head**+**" t:"**+**tail**+**" s:"**+**getSize**()+**"\n"**);**

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

sb**.**append**(**data**[**i**][**0**]+**"\_"**+**data**[**i**][**1**]+**" "**);**

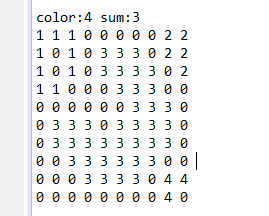
**}**

// sb.append("\n");

**return** sb**.**toString**();**

**}**

**}**



使用dfs染色：

public class Demo **{**

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

int**[][]** book **=** **new** int**[**11**][**11**];**

int**[][]** map **=** **{{**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**},**

**{**0**,**1**,**2**,**1**,**0**,**0**,**0**,**0**,**0**,**2**,**3**},**//1~10

**{**0**,**3**,**0**,**2**,**0**,**1**,**2**,**1**,**0**,**1**,**2**},**

**{**0**,**4**,**0**,**1**,**0**,**1**,**2**,**3**,**2**,**0**,**1**},**

**{**0**,**3**,**2**,**0**,**0**,**0**,**1**,**2**,**4**,**0**,**0**},**

**{**0**,**0**,**0**,**0**,**0**,**0**,**0**,**1**,**5**,**3**,**0**},**

**{**0**,**0**,**1**,**2**,**1**,**0**,**1**,**5**,**4**,**3**,**0**},**

**{**0**,**0**,**1**,**2**,**3**,**1**,**3**,**6**,**2**,**1**,**0**},**

**{**0**,**0**,**0**,**3**,**4**,**8**,**9**,**7**,**5**,**0**,**0**},**

**{**0**,**0**,**0**,**0**,**3**,**7**,**8**,**6**,**0**,**1**,**2**},**

**{**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**,**1**,**0**}};**//

int xBorder **=** map**.**length**-**1**;**

int yBorder **=** map**[**0**].**length**-**1**;**

// printArray2(book);

int color **=** 0**;**

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

**for** **(**int j**=**1**;** j**<=**yBorder**;** j**++)** **{**

**if** **(**map**[**i**][**j**]** **>** 0 **&&** book**[**i**][**j**]** **==** 0**)** **{**

sum **=** 1**;**

drawColor**(**map**,**book**,**i**,** j**,++**color**);**

System**.**out**.**println**(**"sum="**+**sum**+**" color:"**+**color**);**

printArray2**(**book**);**

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

**}**

**}**

**}**

**}**

static int sum **=** 0**;**

**//dfs染色算法**

static void drawColor**(**int**[][]** arr**,**int**[][]** book**,** int x**,** int y**,** int color**)** **{**

int xBorder **=** arr**.**length**-**1**;**

int yBorder **=** arr**[**0**].**length**-**1**;**

book**[**x**][**y**]** **=** color**;**

int**[][]** next **=** **{{**0**,**1**},{**0**,-**1**},{**1**,**0**},{-**1**,**0**}};**

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

int toX **=** x **+** next**[**i**][**0**];**

int toY **=** y **+** next**[**i**][**1**];**

**if** **(**toX **>=**0 **&&** toX **<=** xBorder **&&** toY **>=**0 **&&** toY **<=**yBorder**)** **{**

**if** **(**arr**[**toX**][**toY**]** **>** 0 **&&** book**[**toX**][**toY**]** **==** 0**)** **{**

drawColor**(**arr**,**book**,** toX**,** toY**,** color**);**

sum**++;**

**}**

**}**

**}**

**}**

static void printArray2**(**int**[][]** a**)** **{**

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

**for** **(**int j**=**1**;** j**<**a**[**0**].**length**;** j**++)** **{**

System**.**out**.**printf**(**"%-3d"**,**a**[**i**][**j**]);**

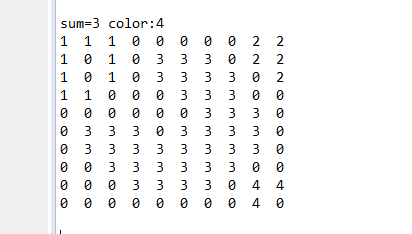
**}**

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

**}**

**}**

**}**



# 图的遍历

public class Main **{**

static int**[][]** map= **{{**0**,**1**,**1**,-**1**,**1**},**

**{**1**,**0**,-**1**,**1**,-**1**},**

**{**1**,-**1**,**0**,-**1**,**1**},**

**{-**1**,**1**,-**1**,**0**,-**1**},**

**{**1**,-**1**,**1**,-**1**,**0**}};**

static int N **=** 5**;**

static int**[]** book **=** **new** int**[**N**+**1**];**

static int sum **=** 0**;**

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

book**[**1**]** **=** 1**;**

dfs**(**1**);**

**}**

//cur 顶点编号

static void dfs**(**int cur**)** **{**

System**.**out**.**printf**(**"%3d"**,**cur**);**

sum**++;**//当前已经遍历过的顶点个数 遍历完毕以后退出

**if** **(**sum **==** N**+**1**)** **return;**

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

**if** **(**map**[**cur**-**1**][**i**-**1**]** **==** 1 **&&** book**[**i**]** **==** 0**)** **{**

book**[**i**]** **=** 1**;**

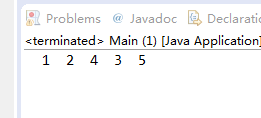
dfs**(**i**);**

**}**

**}**

**}**

**}**



bfs遍历

public class Main **{**

static int**[][]** map **=** **{{**0**,**1**,**1**,-**1**,**1**},**

**{**1**,**0**,-**1**,**1**,-**1**},**

**{**1**,-**1**,**0**,-**1**,**1**},**

**{-**1**,**1**,-**1**,**0**,-**1**},**

**{**1**,-**1**,**1**,-**1**,**0**}};**

static int N **=** 5**;**

static int sum **=** 0**;**

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

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

MyQueue queue **=** **new** MyQueue**();**

queue**.**add**(**1**);**

int cur**=**0**;**

**while** **(!**queue**.**isEmputy**())** **{**

cur **=** queue**.**get**();**

System**.**out**.**println**(**"cur:"**+**cur**);**

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

**if** **(**map**[**cur**-**1**][**i**-**1**]** **==** 1 **&&** book**[**i**]** **==** 0**)** **{**

queue**.**add**(**i**);**

book**[**i**]** **=** 1**;**

**}**

**}**

**}**

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

**}**

static void printArray**(**int**[]** arr**)** **{**

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

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

**}**

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

**}**

**}**

class MyQueue **{**

private static int MAXSIZE **=** 20**;**

private int**[]** data**;**

private int head**;**

private int tail**;**

private int size**;**

public MyQueue**()** **{**

data **=** **new** int**[**MAXSIZE**];**

head **=** 0**;**

tail **=** 0**;**

**}**

public boolean isFull**()** **{**

**if** **((**tail**+**1**)** **%** MAXSIZE **==** head**)**

**return** **true;**

**else**

**return** **false;**

**}**

public boolean isEmputy**()** **{**

**if** **(**tail **==** head**)**

**return** **true;**

**else**

**return** **false;**

**}**

public int getSize**()** **{**

size **=** **(**tail **-** head **+** MAXSIZE**)** **%** MAXSIZE**;**

**return** size**;**

**}**

public boolean add**(**int x**)** **{**

**if** **(**isFull**())** **{**

**return** **false;**

**}** **else** **{**

data**[**tail**]** **=** x**;**

tail **=** **(**tail**+**1**)** **%** MAXSIZE**;**

**return** **true;**

**}**

**}**

public int get**()** **{**

**if** **(**isEmputy**())** **{**

**return** 0**;**

**}** **else** **{**

int e **=** data**[**head**];**

head **=** **(**head**+**1**)** **%** MAXSIZE**;**

**return** e**;**

**}**

**}**

@Override

public String toString**()** **{**

StringBuilder sb **=** **new** StringBuilder**();**

sb**.**append**(**"h:"**+**head**+**" t:"**+**tail**+**" s:"**+**getSize**()+**"\n"**);**

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

sb**.**append**(**data**[**i**]+**" "**);**

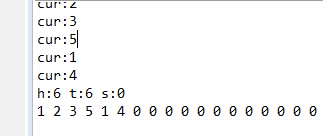
**}**

// sb.append("\n");

**return** sb**.**toString**();**

**}**

**}**



# DFS求最短路径

public class Main **{**

static int**[][]** map **=** **{{**0**,**2**,-**1**,-**1**,**10**},**

**{-**1**,**0**,**3**,-**1**,**7**},**

**{**4**,-**1**,**0**,**4**,-**1**},**

**{-**1**,-**1**,-**1**,**0**,**5**},**

**{-**1**,-**1**,**3**,-**1**,**0**}};**

static int N **=** 5**;**

static int**[]** book **=** **new** int**[**N**+**1**];**

static int sum **=** 0**;**

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

book**[**1**]** **=** 1**;**

dfs**(**1**,** 0**,** 1**);**

System**.**out**.**println**(**"最短路径："**+**minPath**);**

**}**

static int minPath **=** Integer**.**MAX\_VALUE**;**

//cur 顶点编号 ，path路径长度 ，step第几个路过点

static void dfs**(**int cur**,** int path**,** int step**)** **{**

**if** **(**cur **==** 5**)** **{**

**if** **(**minPath **>** path**)** **{**

minPath **=** path**;**

**}**

System**.**out**.**println**(**"路径长度:"**+**path**);**

book**[**5**]** **=** step**;**

System**.**out**.**print**(**"路径："**);**

printArray**(**book**);**

book**[**5**]** **=** 0**;**

**return;**

**}**

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

**if** **(**map**[**cur**-**1**][**i**-**1**]** **>** 0 **&&** book**[**i**]** **==** 0**)** **{**

book**[**i**]** **=** step**;**

dfs**(**i**,** path **+** map**[**cur**-**1**][**i**-**1**],** step**+**1**);**

book**[**i**]** **=** 0**;**

**}**

**}**

**}**

static void printArray**(**int**[]** arr**)** **{**

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

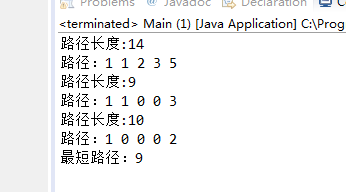
System**.**out**.**print**(**arr**[**i**]+**" "**);**

**}**

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

**}**

**}**



求2点之间的最短路径

public class Main **{**

static int**[][]** map **=** **{{**0**,**2**,**6**,**4**},**

**{-**1**,**0**,**3**,-**1**},**

**{**7**,-**1**,**0**,**1**},**

**{**5**,-**1**,**12**,**0**}**

**};**

static int N **=** 4**;**

static int**[]** book **=** **new** int**[**N**+**1**];**

static int sum **=** 0**;**

static int start **=** 4**;**

static int end **=** 3**;**

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

book**[**start**]** **=** 1**;**

dfs**(**start**,** 0**,** 1**);**

System**.**out**.**println**(**"最短路径："**+**minPath**);**

**}**

static int minPath **=** Integer**.**MAX\_VALUE**;**

//cur 顶点编号 ，path路径长度 ，step第几个路过点

static void dfs**(**int cur**,** int path**,** int step**)** **{**

**if** **(**cur **==** end**)** **{**

**if** **(**minPath **>** path**)** **{**

minPath **=** path**;**

**}**

System**.**out**.**println**(**"路径长度:"**+**path**);**

book**[**end**]** **=** step**;**

System**.**out**.**print**(**"路径："**);**

printArray**(**book**);**

book**[**end**]** **=** 0**;**

**return;**

**}**

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

**if** **(**map**[**cur**-**1**][**i**-**1**]** **>** 0 **&&** book**[**i**]** **==** 0**)** **{**

book**[**i**]** **=** step**;**

dfs**(**i**,** path **+** map**[**cur**-**1**][**i**-**1**],** step**+**1**);**

book**[**i**]** **=** 0**;**

**}**

**}**

**}**

static void printArray**(**int**[]** arr**)** **{**

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

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

**}**

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

**}**

**}**

# Floyd算法求最短路径

public class Main **{**

static int**[][]** map **=** **{{**0**,**2**,**6**,**4**},**

**{**99**,**0**,**3**,**99**},**

**{**7**,**99**,**0**,**1**},**

**{**5**,**99**,**12**,**0**}**

**};**

static int inf **=** 99**;**

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

int row **=** map**.**length**;**

int column **=** map**[**0**].**length**;**

printArray2**(**map**);**

//分别对节点1 到4求最短路径

**for** **(**int k**=**1**;** k**<=**4**;** k**++)** **{**

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

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

**if** **(**map**[**i**][**k**-**1**]** **<** inf **&&** map**[**k**-**1**][**j**]** **<** inf **&&**

map**[**i**][**j**]** **>** map**[**i**][**k**-**1**]+**map**[**k**-**1**][**j**])** **{**

map**[**i**][**j**]** **=** map**[**i**][**k**-**1**]** **+** map**[**k**-**1**][**j**];**

**}**

**}**

**}**

**}**

printArray2**(**map**);**

**}**

static int minPath **=** Integer**.**MAX\_VALUE**;**

static void printArray2**(**int**[][]** arr**)** **{**

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

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

System**.**out**.**printf**(**"%3d"**,**arr**[**i**][**j**]);**

**}**

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

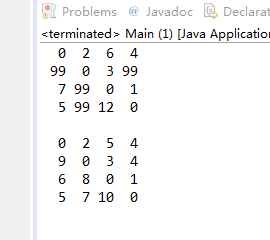
**}**

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

**}**

**}**

}



# Dijkstra算法求最短路径

public class Main **{**

static int**[][]** map **=** **{{**0**,**1**,**12**,**99**,**99**,**99**},**

**{**99**,**0**,**9**,**3**,**99**,**99**},**

**{**99**,**99**,**0**,**99**,**5**,**99**},**

**{**99**,**99**,**4**,**0**,**13**,**15**},**

**{**99**,**99**,**99**,**99**,**0**,**4**},**

**{**99**,**99**,**99**,**99**,**99**,**0**}**

**};**

static int inf **=** 99**;**

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

int points **=** 6**;**//顶点个数

int start **=** 1**;**

int**[]** dis **=** **new** int**[**points**+**1**];**

int**[]** book **=** **new** int**[**points**+**1**];**

//初始化最短距离 start到第i个顶点的距离

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

dis**[**i**]** **=** map**[**start**-**1**][**i**-**1**];**

**}**

printArray**(**dis**);**

book**[**start**]** **=** 1**;**

**for** **(**int k**=**1**;** k**<**points**;** k**++)** **{**

int min **=** inf**,**minId **=** 0**;**

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

**if** **(**book**[**i**]** **==** 0 **&&** min **>** dis**[**i**])** **{**

min **=** dis**[**i**];**

minId **=** i**;**

**}**

**}**

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

book**[**minId**]** **=** 1**;**

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

**if** **(**map**[**minId**-**1**][**i**-**1**]** **<** inf**)** **{**

**if** **(**dis**[**i**]** **>** dis**[**minId**]** **+** map**[**minId**-**1**][**i**-**1**])** **{**

dis**[**i**]** **=** dis**[**minId**]** **+** map**[**minId**-**1**][**i**-**1**];**

**}**

**}**

**}**

printArray**(**dis**);**

**}**

**}**

static int minPath **=** Integer**.**MAX\_VALUE**;**

static void printArray**(**int**[]** arr**)** **{**

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

System**.**out**.**printf**(**"%3d"**,**arr**[**i**]);**

**}**

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

**}**

static void printArray2**(**int**[][]** arr**)** **{**

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

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

System**.**out**.**printf**(**"%3d"**,**arr**[**i**][**j**]);**

**}**

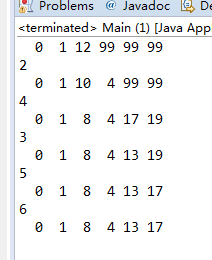
System**.**out**.**println**();**

**}**

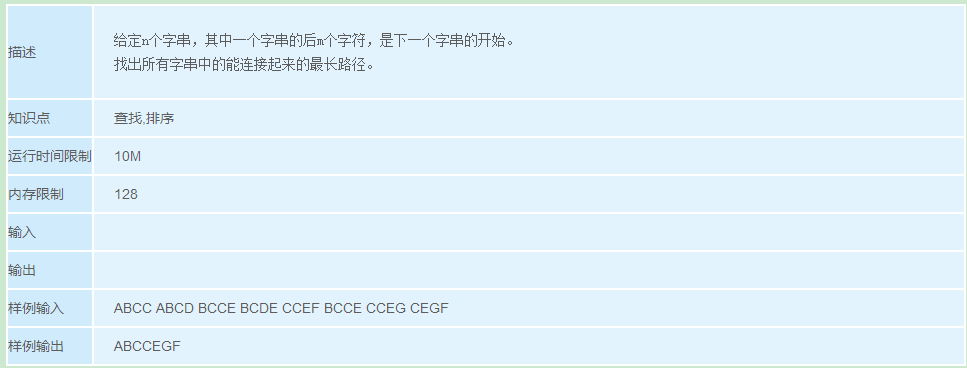
System**.**out**.**println**();**

**}**

**}**



# 字串的连接最长路径查找



package com**.**oj**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String in **=** scanner**.**nextLine**();**

String**[]** strs **=** in**.**split**(**" "**);**

maxCatenate**(**strs**);**

**}**

**}**

static void maxCatenate**(**String**[]** strs**)** **{**

int**[][]** point **=** **new** int**[**strs**.**length**][**strs**.**length**];**

String**[][]** side **=** **new** String**[**strs**.**length**][**strs**.**length**];**

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

String suffix **=** strs**[**i**].**substring**(**1**);**

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

**if** **(**strs**[**j**].**indexOf**(**suffix**)** **==** 0**)** **{**

point**[**i**][**j**]** **=** 1**;**

side**[**i**][**j**]** **=** strs**[**j**].**substring**(**strs**[**j**].**length**()-**1**);**

**}**

**}**

**}**

int len **=** strs**.**length**;**

int maxLen **=** 0**;**

String maxStr **=** ""**;**

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

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

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

**if** **(**point**[**i**][**k**]!=**0 **&&** point**[**k**][**j**]** **!=** 0 **&&**

point**[**i**][**j**]** **<** point**[**i**][**k**]+**point**[**k**][**j**])** **{**

point**[**i**][**j**]** **=** point**[**i**][**k**]+**point**[**k**][**j**];**

side**[**i**][**j**]** **=** side**[**i**][**k**]** **+** side**[**k**][**j**];**

**}**

**}**

**}**

**}**

int max **=** 0**;**

int start **=** 0**;**

int end **=** 0**;**

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

**for** **(**int j **=** 0**;** j **<** strs**.**length**;** j**++)** **{**

**if** **(**max **<** point**[**i**][**j**]** **&&** i **!=** j**)** **{**

max **=** point**[**i**][**j**];**

start **=** i**;**

end **=** j**;**

**}**

**}**

**}**

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

System**.**out**.**println**(**strs**[**start**]** **+** side**[**start**][**end**]);**

**}**

**}**

static void printArray**(**int**[]** arr**)** **{**

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

System**.**out**.**printf**(**"%3d"**,**arr**[**i**]);**

**}**

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

**}**

static void printArray2**(**int**[][]** arr**)** **{**

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

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

System**.**out**.**printf**(**"%3d"**,**arr**[**i**][**j**]);**

**}**

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

**}**

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

**}**

static void printStrs**(**String**[][]** arr**)** **{**

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

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

System**.**out**.**printf**(**"%5s"**,**arr**[**i**][**j**]);**

**}**

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

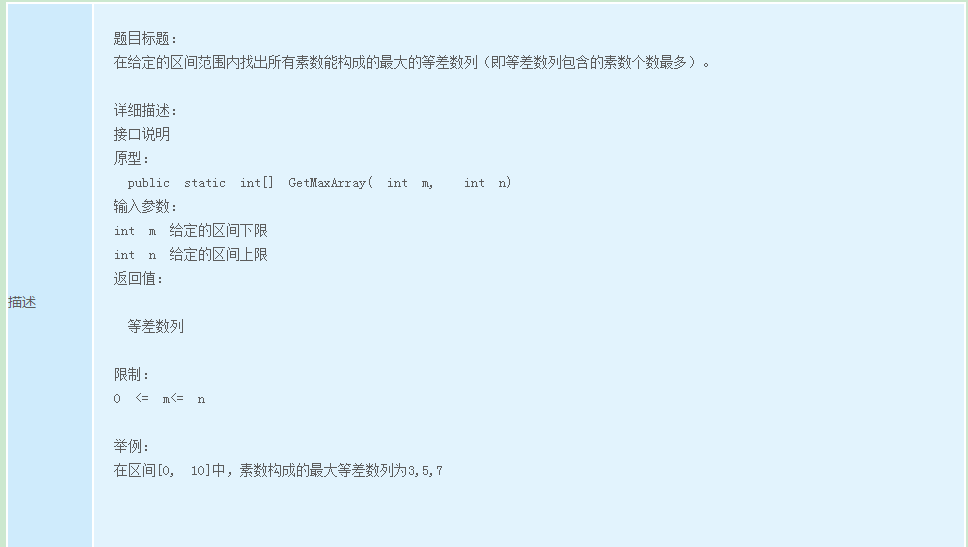
**}**

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

**}**

**}**

# 寻找等差数列



<http://www.lxway.com/814214291.htm>

<http://www.1point3acres.com/bbs/thread-14078-1-1.html>

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

int start **=** scanner**.**nextInt**();**

int end **=** scanner**.**nextInt**();**

ArrayList**<**Integer**>** primes **=** **new** ArrayList**<>();**

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

**if** **(**isPrime**(**i**))** **{**

primes**.**add**(**i**);**

**}**

**}**

// System.out.println(primes);

find2**(**primes**);**

**}**

scanner**.**close**();**

**}**

public static void find2**(**List**<**Integer**>** primes**)** **{**

int len **=** primes**.**size**();**

int**[][]** dp **=** **new** int**[**len**][**len**];**

//初始化数组

// dp[i][j]表示以a[i]和a[j]结尾的最长等差数列的长度。

// 枚举最后两个元素，对于每一个a[i]和a[j]，

// 都要找到a[p]，即a[p],a[i],a[j]; p < i，满足a[p] + a[j] == 2 \* a[i]。

// 然后dp[p][i] + 1去更新dp[i][j]

//初始化

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

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

dp**[**m**][**n**]** **=** 2**;**

**}**

**}**

int maxLen **=** 0**,** maxI**=**0**,**maxJ**=**0**;**

**for** **(**int j**=**1**;** j**<**len**;** j**++)** **{**

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

int p **=** 0**;**

**while** **(**p**<**i**)** **{**

**if** **((**primes**.**get**(**p**)+**primes**.**get**(**j**))** **==** 2**\***primes**.**get**(**i**))** **{**

dp**[**i**][**j**]** **=** dp**[**p**][**i**]** **+** 1**;**

**break;**

**}**

p**++;**

**}**

**if** **(**maxLen **<=** dp**[**i**][**j**])** **{**

maxLen **=** dp**[**i**][**j**];**

maxI **=** i**;**

maxJ **=** j**;**

**}**

**}**

**}**

// printArray2(dp);

int tail **=** primes**.**get**(**maxJ**);**

int diff **=** primes**.**get**(**maxJ**)-**primes**.**get**(**maxI**);**

**for** **(**int i**=**maxLen**-**1**;** i**>=**0**;** i**--)** **{**

System**.**out**.**println**(**tail**-**diff**\***i**);**

**}**

**}**

private static void find**(**ArrayList**<**Integer**>** primes**)** **{**

int maxDiff **=** primes**.**get**(**primes**.**size**()-**1**)** **-** primes**.**get**(**0**);**

int size **=** primes**.**size**();**

//dp[i][d]表示以第i个元素为结尾，以d为等差的最长等差数列的长度。

//dp[i][d] = dp[j][d] + 1; (j<=i-1,d=num[i]-num[j])

int**[][]** dp **=** **new** int**[**size**][**maxDiff**+**1**];**

//初始化数组

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

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

dp**[**m**][**n**]** **=** 1**;**

**}**

**}**

int maxLen **=** 0**,**maxM**=**0**,**maxN**=**0**;**

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

**for** **(**int n**=**m**-**1**;** n**>=**0**;** n**--)** **{**

int dif **=** primes**.**get**(**m**)-**primes**.**get**(**n**);**

dp**[**m**][**dif**]** **=** dp**[**n**][**dif**]** **+** 1**;**

//maxLen表示等差数列的长度

//maxM第几个元素

//maxN表示等差值

**if** **(**maxLen **<=** dp**[**m**][**dif**])** **{**

maxLen **=** dp**[**m**][**dif**];**

maxM **=** m**;**

maxN **=** dif**;**

**}**

**}**

**}**

// System.out.println(maxDiff);

// printArray2(dp);

**for** **(**int i**=**maxLen**-**1**;** i**>=**0**;** i**--)** **{**

System**.**out**.**println**(**primes**.**get**(**maxM**)-**i**\***maxN**);**

**}**

**}**

static boolean isPrime**(**int n**)** **{**

**if** **(**n **<** 2**)** **{**

**return** **false;**

**}**

int end **=** **(**int**)** Math**.**sqrt**(**n**);**

int i**;**

**for** **(**i**=**2**;** i**<=**end**;** i**++)** **{**

**if** **(**n **%** i **==** 0**)**

**break;**

**}**

**if** **(**i **>** end**)** **{**

**return** **true;**

**}** **else** **{**

**return** **false;**

**}**

**}**

static void printArray**(**int**[]** arr**)** **{**

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

System**.**out**.**printf**(**"%3d"**,**arr**[**i**]);**

**}**

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

**}**

static void printArray2**(**int**[][]** arr**)** **{**

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

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

System**.**out**.**printf**(**"%3d"**,**arr**[**i**][**j**]);**

**}**

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

**}**

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

**}**

**}**

# 密码强度等级

密码按如下规则进行计分，并根据不同的得分为密码进行安全等级划分。

       一、密码长度:

       5 分: 小于等于4 个字符

       10 分: 5 到7 字符

       25 分: 大于等于8 个字符

       二、字母:

       0 分: 没有字母

       10 分: 全都是小（大）写字母

       20 分: 大小写混合字母

       三、数字:

       0 分: 没有数字

       10 分: 1 个数字

       20 分: 大于1 个数字

       四、符号:

       0 分: 没有符号

       10 分: 1 个符号

       25 分: 大于1 个符号

       五、奖励:

       2 分: 字母和数字

       3 分: 字母、数字和符号

       5 分: 大小写字母、数字和符号

       最后的评分标准:

       >= 90: 非常安全

       >= 80: 安全（Secure）

       >= 70: 非常强

       >= 60: 强（Strong）

       >= 50: 一般（Average）

       >= 25: 弱（Weak）

       >= 0:  非常弱

对应输出为：

  VERY\_WEAK,

   WEAK,

   AVERAGE,

   STRONG,

   VERY\_STRONG,

   SECURE,

   VERY\_SECURE

       请根据输入的密码字符串，进行安全评定。

       注：

       字母：a-z, A-Z

       数字：-9

       符号包含如下： (ASCII码表可以在UltraEdit的菜单view->ASCII Table查看)

       !"#$%&'()\*+,-./     (ASCII码：x21~0x2F)

       :;<=>?@             (ASCII<=><=><=><=><=>码：x3A~0x40)

       [\]^\_`              (ASCII码：x5B~0x60)

  {|}~                (ASCII码：x7B~0x7E)

package com**.**oj**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String in **=** scanner**.**nextLine**();**

int alpha **=** getAlpahaScore**(**in**);**

int digit **=** getDigitScore**(**in**);**

int symbol **=** getSymbolScore**(**in**);**

int len **=** getLenScore**(**in**);**

int other **=** getOtherScore**(**alpha**,** digit**,** symbol**);**

int total **=** alpha**+**digit**+**symbol**+**other**+**len**;**

**if** **(**total **>=** 90**)** **{**

System**.**out**.**println**(**"VERY\_SECURE"**);**

**}** **else** **if** **(**total **>=** 80**)** **{**

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

**}** **else** **if** **(**total **>=** 70**)** **{**

System**.**out**.**println**(**"VERY\_STRONG"**);**

**}** **else** **if** **(**total **>=** 60**)** **{**

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

**}** **else** **if** **(**total **>=** 50**)** **{**

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

**}** **else** **if** **(**total **>=** 25**)** **{**

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

**}** **else** **if** **(**total **>=** 0**)** **{**

System**.**out**.**println**(**"VERY\_WEAK"**);**

**}**

**}**

scanner**.**close**();**

**}**

static int getLenScore**(**String str**)** **{**

int len **=** str**.**length**();**

int score **=** 0**;**

**if** **(**len **>=** 8**)** **{**

score **=** 25**;**

**}** **else** **if** **(**len **>=** 5**){**

score **=** 10**;**

**}** **else{**

score **=** 5**;**

**}**

**return** score**;**

**}**

static int getAlpahaScore**(**String str**)** **{**

int up**=**0**,**down**=**0**,**score**=**0**;**

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

char ch **=** str**.**charAt**(**i**);**

**if** **(**ch**>=**'a' **&&** ch**<=**'z'**)** **{**

down**++;**

**}** **else** **if** **(**ch**>=**'A' **&&** ch**<=**'Z'**)** **{**

up**++;**

**}**

**}**

**if** **(**up**==**0 **&&** down**==**0**)** **{**

score **=** 0**;**

**}** **else** **if** **(**up**==**0 **||** down**==**0**){**

score **=** 10**;**

**}** **else** **{**

score **=** 20**;**

**}**

**return** score**;**

**}**

static int getDigitScore**(**String str**)** **{**

int times**=**0**,**score**=**0**;**

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

**if** **(**str**.**charAt**(**i**)** **>=** '0' **&&** str**.**charAt**(**i**)** **<=** '9'**)** **{**

times**++;**

**}**

**}**

**if** **(**times **==** 0**)** **{**

score **=** 0**;**

**}** **else** **if** **(**times **==** 1**)** **{**

score **=** 10**;**

**}** **else** **{**

score **=** 20**;**

**}**

**return** score**;**

**}**

static int getSymbolScore**(**String str**)** **{**

int times**=**0**,** score**=**0**;**

int ch**;**

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

ch **=** str**.**charAt**(**i**);**

**if** **((**ch**>=**0x21**&&**ch**<=**0x2f**)||(**ch**>=**0x3a**&&**ch**<=**0x40**)**

**||(**ch**>=**0x5b**&&**ch**<=**0x60**)||(**ch**>=**0x7b**&&**ch**<=**0x7e**))** **{**

times**++;**

**}**

**}**

**if** **(**times **==** 0**)** **{**

score **=** 0**;**

**}** **else** **if** **(**times **==** 1**)** **{**

score **=** 10**;**

**}** **else** **{**

score **=** 20**;**

**}**

**return** score**;**

**}**

static int getOtherScore**(**int alpha**,** int digit**,** int symbol**)** **{**

int score**=**0**;**

**if** **(**alpha**==**20 **&&** digit**>=**10 **&&** symbol**>=**10**)** **{**

score **=** 5**;**

**}** **else** **if** **(**alpha**==**10 **&&** digit**>=**10 **&&** symbol**>=**10**)** **{**

score **=** 3**;**

**}** **else** **if** **(**alpha **>=**10 **&&** digit**>=**10**)** **{**

score **=** 2**;**

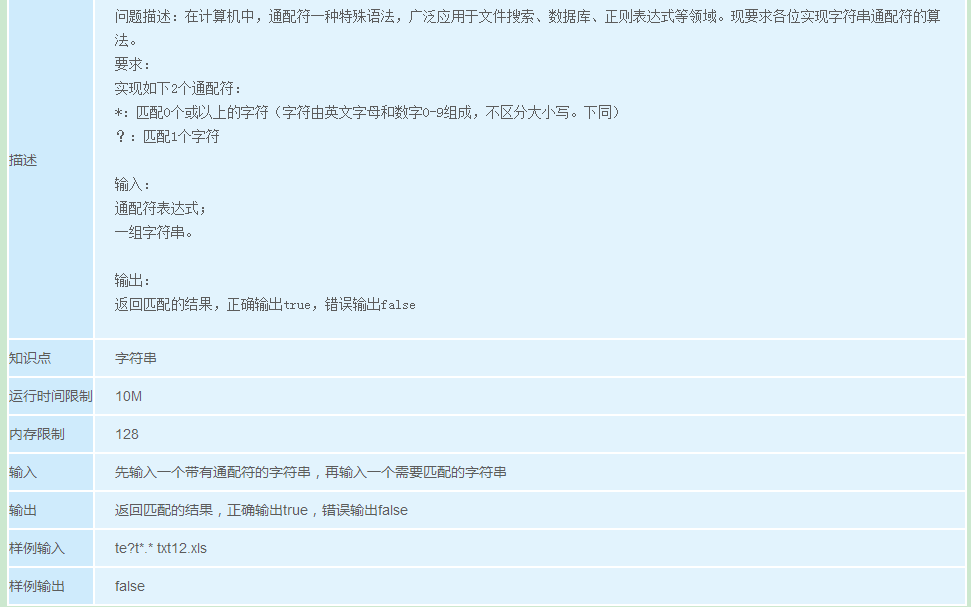
**}**

**return** score**;**

**}**

**}**

# 字符串通配符



package com**.**oj**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String regex **=** scanner**.**next**().**toLowerCase**();**

String text **=** scanner**.**next**().**toLowerCase**();**

System**.**out**.**println**(**matchHere**(**text**,** regex**));**

**}**

scanner**.**close**();**

**}**

//逐个匹配

static boolean matchHere**(**String text**,** String regex**)** **{**

//当匹配字符串结束

**if** **(**regex**.**length**()** **==** 0**)** **{**

**return** **(**text**.**length**()** **==** 0**);**

**}**

//遇到\*号

**if** **(**regex**.**charAt**(**0**)** **==** '\*'**)** **{**

**return** matchStart**(**text**,** regex**.**substring**(**1**));**

**}**

//当被匹配字符串结束，判断匹配字符串的下一个是否为\*号

//是则继续匹配，不是则匹配失败

//如 abc abc\*; abc abc\*d; abc abcd

**if** **(**text**.**length**()** **==** 0**)** **{**

**if** **(**regex**.**charAt**(**0**)** **==** '\*'**)**

**return** matchHere**(**text**,** regex**);**

**else**

**return** **false;**

**}**

//如果2个字符串第一个字符相等 或者匹配字符为？号，则继续匹配后面的字符串

**if** **((**text**.**charAt**(**0**)** **==** regex**.**charAt**(**0**))**

**||** regex**.**charAt**(**0**)==**'?'**)** **{**

**return** matchHere**(**text**.**substring**(**1**),** regex**.**substring**(**1**));**

**}**

**return** **false;**

**}**

//匹配星号

static boolean matchStart**(**String text**,** String regex**)** **{**

//表明当前只剩一个\*号，可以匹配

**if** **(**regex**.**length**()** **==** 0**)** **{**

**return** **true;**

**}**

// do {

// System.out.println(text);

// if (matchH(text, regex)) {

// return true;

// }

// text = text.substring(1);

// } while(text.length() > 0);

**while(**text**.**length**()** **>** 0**)** **{**

**if** **(**matchHere**(**text**,** regex**))** **{**

**return** **true;**

**}**

//对\*号后的字符匹配比较，如果不匹配则跳过

//如 abc和\*c 依次比较abc c->bc c->c c->true

text **=** text**.**substring**(**1**);**

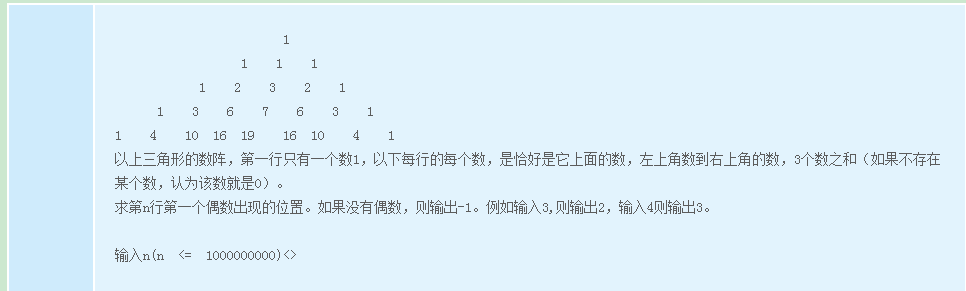
**}**

**return** **false;**

**}**

**}**

# 杨辉三角的变形

package com**.**oj**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

int row **=** 0**;**

row **=** scanner**.**nextInt**();**

**for** **(**int j**=**1**;** j**<**2**\***row**;** j**++)** **{**

**if** **(**getPoint**(**row**,** j**)** **%** 2 **==** 0**)** **{**

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

**break;**

**}**

**if** **(**j **==** **(**2**\***row**-**1**))** **{**

System**.**out**.**println**(-**1**);**

**}**

**}**

**}**

scanner**.**close**();**

**}**

static long getPoint**(**long row**,** long column**)** **{**

**if** **(**column **==** 1**)**

**return** 1**;**

**if** **(**column **<** 1 **||** column **>** **(**2**\***row**-**1**))** **{**

**return** 0**;**

**}**

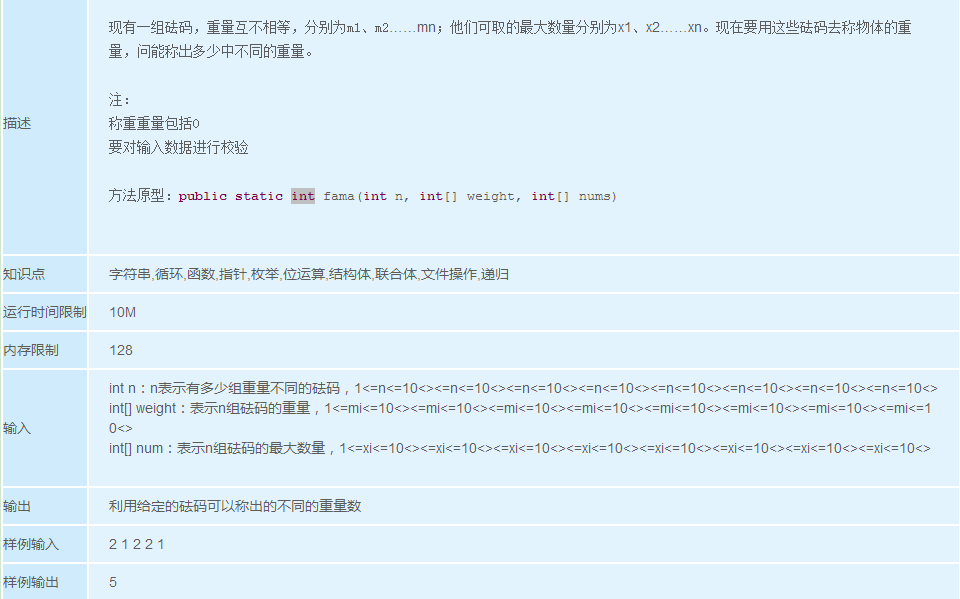
**return** getPoint**(**row**-**1**,** column**-**2**)+**getPoint**(**row**-**1**,** column**-**1**)**

**+**getPoint**(**row**-**1**,** column**);**

**}**

**}**

# 称砝码



package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**HashSet**;**

**import** java**.**util**.**Iterator**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

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

int**[]** weight **=** **new** int**[**n**+**1**];**

int**[]** count **=** **new** int**[**n**+**1**];**

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

weight**[**i**]** **=** scanner**.**nextInt**();**

**}**

int maxWeight **=** 0**;**

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

count**[**i**]** **=** scanner**.**nextInt**();**

maxWeight **+=** weight**[**i**]\***count**[**i**];**

**}**

// find(weight, count, maxWeight);

find2**(**weight**,** count**);**

**}**

scanner**.**close**();**

**}**

private static void find**(**int**[]** weight**,** int**[]** count**,** int maxWeight**)** **{**

int n **=** weight**.**length**-**1**;**

int**[]** dp **=** **new** int**[**maxWeight**+**1**];**

dp**[**0**]** **=** 1**;**

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

**for** **(**int m **=** maxWeight**;** m**>=**0**;** m**--)** **{**

**for** **(**int j**=**1**;** j**<=**count**[**i**];** j**++)** **{**

dp**[**m**]** **=** dp**[**m**]** **|** **((**m**-**j**\***weight**[**i**]** **<** 0**)?**0**:**dp**[**m**-**j**\***weight**[**i**]]);**

**}**

**}**

**}**

// print(dp);

int total**=**0**;**

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

total **+=** dp**[**i**];**

**}**

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

**}**

private static void find2**(**int**[]** weight**,** int**[]** count**)** **{**

int n **=** weight**.**length**-**1**;**

ArrayList**<**Integer**>** weights **=** **new** ArrayList**<>();**

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

**for** **(**int j**=**1**;** j**<=**count**[**i**];** j**++)** **{**

weights**.**add**(**weight**[**i**]);**

**}**

**}**

// System.out.println(weights);

HashSet**<**Integer**>** ws **=** **new** HashSet**<>();**

ws**.**add**(**0**);**

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

int w **=** weights**.**get**(**i**);**

Iterator**<**Integer**>** it **=** ws**.**iterator**();**

HashSet**<**Integer**>** temp **=** **new** HashSet**<>(**ws**);**

**while** **(**it**.**hasNext**())** **{**

temp**.**add**(**w**+**it**.**next**());**

**}**

ws **=** temp**;**

**}**

System**.**out**.**println**(**ws**.**size**());**

**}**

static void print**(**int**[]** a**)** **{**

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

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

**}**

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

**}**

**}**

# 合唱队



package com**.**oj**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

int n**;**

n **=** scanner**.**nextInt**();**

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

//input height

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

height**[**i**]** **=** scanner**.**nextInt**();**

**}**//for

getPeople**(**height**);**

scanner**.**close**();**

**}**

static void getPeople**(**int**[]** height**)** **{**

int len **=** height**.**length**;**

//从左到右的递增子序列最长长度

//lis[i]表示以第i个元素为结尾 的数列中的 最长递增子序列的长度

int**[]** lis **=** **new** int**[**len**];**

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

lis**[**i**]** **=** 1**;**

**}**

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

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

**if** **(**height**[**n**]** **<** height**[**m**]** **&&** lis**[**m**]** **<** **(**lis**[**n**]** **+** 1**))** **{**

lis**[**m**]** **=** lis**[**n**]** **+** 1**;**

**}**

**}**

**}**

// print(lis);

//从右到左的递增子序列最长长度

int**[]** lds **=** **new** int**[**len**];**

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

lds**[**i**]** **=** 1**;**

**}**

**for** **(**int m**=**len**-**2**;** m**>=**0**;** m**--)** **{**

**for** **(**int n**=**len**-**1**;** n**>**m**;** n**--)** **{**

**if** **(**height**[**m**]** **>** height**[**n**]** **&&** lds**[**m**]** **<** **(**lds**[**n**]** **+** 1**))** **{**

lds**[**m**]** **=** lds**[**n**]** **+** 1**;**

**}**

**}**

**}**

// print(lds);

int maxLen **=** 0**;**

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

**if** **(**maxLen **<** **(**lis**[**i**]** **+** lds**[**i**])-**1**)** **{**

maxLen **=** **(**lis**[**i**]** **+** lds**[**i**])-**1**;**

**}**

**}**

System**.**out**.**println**(**len **-** maxLen**);**

**}**

static void print**(**int**[]** a**)** **{**

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

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

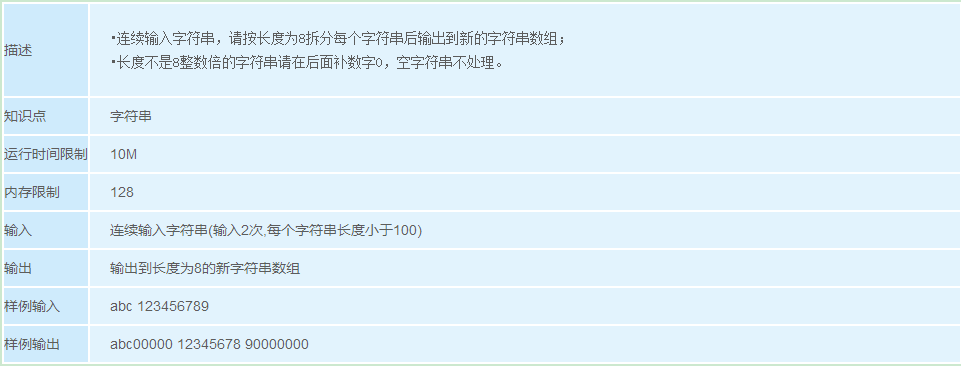
**}**

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

**}**

**}**

# 字符串分隔

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String in **=** scanner**.**nextLine**();**

String**[]** ins **=** in**.**split**(**" "**);**

ArrayList**<**String**>** list **=** **new** ArrayList**<>();**

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

splitStr**(**list**,** ins**[**i**]);**

**}**

// System.out.println(list);

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

System**.**out**.**println**(**list**.**get**(**i**));**

**}**

**}**

scanner**.**close**();**

**}**

static void splitStr**(**ArrayList**<**String**>** list**,** String str**)** **{**

int len **=** str**.**length**();**

**if** **(**len **>** 8**)** **{**

int id **=** 0**;**

String part**;**

**while** **(**id **+** 8 **<=** len**)** **{**

part **=** str**.**substring**(**id**,** id**+**8**);**

id **+=** 8**;**

list**.**add**(**part**);**

**}**

**if** **(**id **<** len**)** **{**

list**.**add**(**getEightStr**(**str**.**substring**(**id**,**len**)));**

**}**

**}** **else** **{**

list**.**add**(**getEightStr**(**str**));**

**}**

**}**

static String getEightStr**(**String str**)** **{**

int len **=** str**.**length**();**

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

**if** **(**len **<=** 8**)** **{**

int i **=** 8 **-** len**;**

**while** **(**i**--** **>** 0**)** **{**

sb**.**append**(**'0'**);**

**}**

**}**

**return** sb**.**toString**();**

**}**

**}**

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Scanner;

**public** **class** Main {

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

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

**while** (scanner.hasNext()) {

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

List<String> list = **new** ArrayList<>();

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

*splitStr*(list, scanner.next());

}

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

System.***out***.println(list.get(i));

}

}

scanner.close();

}

**static** **void** splitStr(List<String> list, String str) {

**if** (str.length() > 8) {

**int** id = 0;

**while** (id + 8 <= str.length()) {

list.add(str.substring(id, id+8));

id += 8;

}

**if** (id < str.length()) {

list.add(*get8Str*(str.substring(id, str.length())));

}

} **else** {

list.add(*get8Str*(str));

}

}

**static** String get8Str(String str) {

StringBuilder sb = **null**;

**if** (str.length() <= 8) {

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

**int** len = str.length();

**int** i = 8 - len;

**while** (i > 0) {

sb.append('0');

i--;

}

} **else** {

**return** **null**;

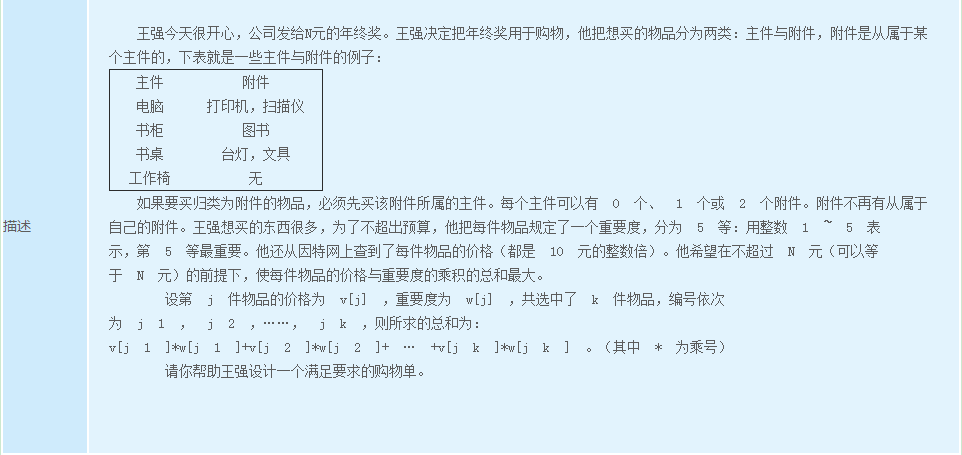
}

**return** sb.toString();

}

}

# 购物单



package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

int money **=** scanner**.**nextInt**()/**10**;**

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

ArrayList**<**Goods**>** list **=** **new** ArrayList**<>();**

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

int v **=** scanner**.**nextInt**()/**10**;**

int w **=** scanner**.**nextInt**();**

int q **=** scanner**.**nextInt**();**

**if** **(**q **==** 0**)** **{**

//添加主件,其编号为下标+1

Goods goods **=** **new** Goods**(**v**,** w**);**

list**.**add**(**goods**);**

**}** **else** **{**

//添加附件，q为主件编号，对应在第q-1位置的元素

list**.**get**(**q**-**1**).**addChild**(**v**,** w**);**

**}**

**}**

int len **=** list**.**size**();**

//dp[i][j]表示对于前i个物品，使用j元钱购买

//所能获取的最大价格与重要度的乘积的总和

//dp[i][j] = max(dp[i-1][j](没买),dp[i-1][j-v]+v\*w（买入）);

int**[][]** dp **=** **new** int**[**len**+**1**][**money**+**1**];**

int v **=** 0**,** w **=** 0**,** v1 **=** 0**,** w1 **=** 0**,** v2 **=** 0**,** w2 **=** 0**,** max **=** 0**;**

Goods goods**;**

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

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

goods **=** list**.**get**(**i**-**1**);**

//只计算主件

v **=** goods**.**value**;**

w **=** goods**.**weight**;**

**if** **(**j **>=** v**)** **{**

dp**[**i**][**j**]** **=** max**(**dp**[**i**-**1**][**j**],**dp**[**i**-**1**][**j**-**v**]+**v**\***w**);**

**if** **(**max **<** dp**[**i**][**j**])** **{**

max **=** dp**[**i**][**j**];**

**}**

**}**

//计算主件和第一个附件

**if** **(**goods**.**child1 **!=** **null)** **{**

v1 **=** goods**.**child1**.**value**;**

w1 **=** goods**.**child1**.**weight**;**

**if** **(**j **>=** v**+**v1**)** **{**

dp**[**i**][**j**]** **=** max**(**dp**[**i**-**1**][**j**]**

**,**dp**[**i**-**1**][**j**-**v**-**v1**]** **+** v**\***w **+** v1**\***w1**);**

**if** **(**max **<** dp**[**i**][**j**])** **{**

max **=** dp**[**i**][**j**];**

**}**

**}**

**}**

//计算主件和第二个附件

**if** **(**goods**.**child2 **!=** **null)** **{**

v2 **=** goods**.**child2**.**value**;**

w2 **=** goods**.**child2**.**weight**;**

**if** **(**j **>=** v**+**v2**)** **{**

dp**[**i**][**j**]** **=** max**(**dp**[**i**-**1**][**j**]**

**,**dp**[**i**-**1**][**j**-**v**-**v2**]** **+** v**\***w **+** v2**\***w2**);**

**if** **(**max **<** dp**[**i**][**j**])** **{**

max **=** dp**[**i**][**j**];**

**}**

**}**

//计算主件和第一、第二个附件

**if** **(**j **>=** v**+**v1**+**v2**)** **{**

dp**[**i**][**j**]** **=** max**(**dp**[**i**-**1**][**j**]**

**,**dp**[**i**-**1**][**j**-**v**-**v1**-**v2**]** **+** v**\***w **+**v1**\***w1 **+** v2**\***w2**);**

**if** **(**max **<** dp**[**i**][**j**])** **{**

max **=** dp**[**i**][**j**];**

**}**

**}**

**}**

**}**

**}**

// System.out.println(list);

// printArray2(dp);

System**.**out**.**println**(**max**\***10**);**

**}**

scanner**.**close**();**

**}**

private static int max**(**int i**,** int j**)** **{**

// TODO Auto-generated method stub

**return** i **>** j **?** i **:** j**;**

**}**

static void printArray2**(**int**[][]** ns**)** **{**

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

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

System**.**out**.**printf**(**"%-3d "**,**ns**[**i**][**j**]);**

**}**

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

**}**

**}**

**}**

class Goods **{**

public int value**;**

public int weight**;**

Goods child1**;**

Goods child2**;**

public Goods**(**int v**,** int w**)** **{**

value **=** v**;**

weight **=** w**;**

**}**

//只能依次添加附件1，附件2

public void addChild**(**int v**,** int w**)** **{**

**if** **(**child1 **==** **null)** **{**

child1 **=** **new** Goods**(**v**,** w**);**

**}** **else** **if** **(**child2 **==** **null)** **{**

child2 **=** **new** Goods**(**v**,** w**);**

**}**

**}**

public String toString**()** **{**

StringBuilder sb **=** **new** StringBuilder**();**

sb**.**append**(**"v:"**+**value**+**" w:"**+**weight**);**

**if** **(**child1 **!=** **null)** **{**

sb**.**append**(**" child1:"**+**child1**.**value**+**" "**+**child1**.**weight**);**

**}**

**if** **(**child2 **!=** **null)** **{**

sb**.**append**(**" child2:"**+**child2**.**value**+**" "**+**child2**.**weight**);**

**}**

**return** sb**.**toString**();**

**}**

**}**

# 识别有效的IP地址和掩码并进行分类统计

|  |  |
| --- | --- |
| 描述 | 请解析IP地址和对应的掩码，进行分类识别。  要求按照A/B/C/D/E类地址归类，不合法的地址和掩码单独归类。    所有的IP地址划分为 A,B,C,D,E五类    A类地址1.0.0.0~126.255.255.255;    B类地址128.0.0.0~191.255.255.255;    C类地址192.0.0.0~223.255.255.255;    D类地址224.0.0.0~239.255.255.255；    E类地址240.0.0.0~255.255.255.255        私网IP范围是：    10.0.0.0～10.255.255.255    172.16.0.0～172.31.255.255    192.168.0.0～192.168.255.255        子网掩码为前面是连续的1，然后全是0 |
| 知识点 | 字符串,循环,查找,搜索,排序,函数,指针,枚举,位运算,结构体,联合体,文件操作,递归 |
| 运行时间限制 | 0M |
| 内存限制 | 0 |
| 输入 | 多行字符串。每行一个IP地址和掩码，已~隔开。如：    10.70.44.68~255.254.255.0    1.0.0.1~255.0.0.0    192.168.0.2~255.255.255.0    19..0.~255.255.255.0 |
| 输出 | 统计A、B、C、D、E、错误IP地址或错误掩码、私有IP的个数，之间以空格隔开，根据上面的IP，可以得到：    1.0.0.1~255.0.0.0 ----A类    192.168.0.2~255.255.255.0  ----C类，私有    10.70.44.68~255.254.255.0----错误的掩码    19..0.~255.255.255.0-----错误的IP    可以得到统计数据如下：    1 0 1 0 0 2 1 |

样例输入

10.70.44.68~255.254.255.0

1.0.0.1~255.0.0.0

192.168.0.2~255.255.255.0

19..0.~255.255.255.0

|  |  |
| --- | --- |
| 样例输出 | 1 0 1 0 0 2 1 |

package com**.**oj**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

int**[]** types **=** **new** int**[**8**];**

**while** **(**scanner**.**hasNextLine**())** **{**

String in **=** scanner**.**nextLine**();**

// if (in.length()==0) break;

String**[]** ins **=** in**.**split**(**"\\~"**);**

int id **=** getType**(**ins**[**0**],** ins**[**1**]);**

types**[**id**]++;**

**if** **(**isPrivate**(**ins**[**0**],** ins**[**1**]))** **{**

types**[**7**]++;**

**}**

**}**

**for** **(**int i**=**1**;** i**<**8**;** i**++)** **{**

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

**}**

scanner**.**close**();**

**}**

//1-6分别代表A、B、C、D、E、错误

static int getType**(**String ip**,** String mask**)** **{**

**if** **(**isValidIP**(**ip**)** **&&** isValidMask**(**mask**))** **{**

String**[]** ips **=** ip**.**split**(**"\\."**);**

//判断类似12..0.1

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

**if** **(**ips**[**i**].**length**()** **==** 0**)**

**return** 6**;**

**}**

int ip1 **=** Integer**.**parseInt**(**ips**[**0**]);**

int ip2 **=** Integer**.**parseInt**(**ips**[**1**]);**

String**[]** masks **=** mask**.**split**(**"\\."**);**

int m1 **=** Integer**.**parseInt**(**masks**[**0**]);**

int m2 **=** Integer**.**parseInt**(**masks**[**1**]);**

int h1 **=** ip1 **&** m1**;**

int h2 **=** ip2 **&** m2**;**

**if** **(**h1 **>=** 1 **&&** h1 **<=** 126**)** **{**

**return** 1**;**

**}** **else** **if** **(**h1 **>=** 128 **&&** h1 **<=** 191**)** **{**

**return** 2**;**

**}** **else** **if** **(**h1 **>=** 192 **&&** h1 **<=** 223**)** **{**

**return** 3**;**

**}** **else** **if** **(**h1 **>=** 224 **&&** h1 **<=** 239**)** **{**

**return** 4**;**

**}** **else** **if** **(**h1 **>=** 240 **&&** h1 **<=** 255**)** **{**

**return** 5**;**

**}**

**if** **(**h1 **==** 10 **||** **(**h1 **==** 192 **&&** h2 **==** 168**))** **{**

**return** 7**;**

**}** **else** **if** **(**h1 **==** 172 **&&** **(**h2**>=**16 **&&** h2**<=**31**))** **{**

**return** 7**;**

**}**

**}** **else** **{**

**return** 6**;**

**}**

**return** 0**;**

**}**

//私有单独判断，包含在A B C 类中

static boolean isPrivate**(**String ip**,** String mask**)** **{**

**if** **(**isValidIP**(**ip**)** **&&** isValidMask**(**mask**))** **{**

String**[]** ips **=** ip**.**split**(**"\\."**);**

//判断类似12..0.1

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

**if** **(**ips**[**i**].**length**()** **==** 0**)**

**return** **false;**

**}**

int ip1 **=** Integer**.**parseInt**(**ips**[**0**]);**

int ip2 **=** Integer**.**parseInt**(**ips**[**1**]);**

String**[]** masks **=** mask**.**split**(**"\\."**);**

int m1 **=** Integer**.**parseInt**(**masks**[**0**]);**

int m2 **=** Integer**.**parseInt**(**masks**[**1**]);**

int h1 **=** ip1 **&** m1**;**

int h2 **=** ip2 **&** m2**;**

**if** **(**h1 **==** 10 **||** **(**h1 **==** 192 **&&** h2 **==** 168**))** **{**

**return** **true;**

**}** **else** **if** **(**h1 **==** 172 **&&** **(**h2**>=**16 **&&** h2**<=**31**))** **{**

**return** **true;**

**}**

**}** **else** **{**

**return** **false;**

**}**

**return** **false;**

**}**

static boolean isValidIP**(**String ip**)** **{**

String regex **=** "(25[0-5]||2[0-4][0-9]||1[0-9][0-9]||[1-9][0-9]||[1-9])"

**+** "\\.(25[0-5]||2[0-4][0-9]||[0-1]?[0-9]?[0-9])"

**+** "\\.(25[0-5]||2[0-4][0-9]||[0-1]?[0-9]?[0-9])"

**+** "\\.(25[0-5]||2[0-4][0-9]||[0-1]?[0-9]?[0-9])"**;**

//有缺陷 无法判断类似12..0.1

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

// if (ips[i].length() == 0)

// return false;

// }

**return** ip**.**matches**(**regex**);**

**}**

static boolean isValidMask**(**String mask**)** **{**

String regex **=** "(25[0-5]||2[0-4][0-9]||[0-1]?[0-9]?[0-9])"

**+** "\\.(25[0-5]||2[0-4][0-9]||[0-1]?[0-9]?[0-9])"

**+** "\\.(25[0-5]||2[0-4][0-9]||[0-1]?[0-9]?[0-9])"

**+** "\\.(25[0-5]||2[0-4][0-9]||[0-1]?[0-9]?[0-9])"**;**

**if** **(**mask**.**matches**(**regex**))** **{**

String**[]** ms **=** mask**.**split**(**"\\."**);**

StringBuilder sb **=** **new** StringBuilder**();**

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

sb**.**append**(**getBin8**(**ms**[**i**]));**

**}**

// System.out.println(sb.toString());

**if** **(**sb**.**toString**().**matches**(**"^[1]+[0]+$"**))** **{**

**return** **true;**

**}** **else** **{**

**return** **false;**

**}**

**}** **else** **{**

**return** **false;**

**}**

**}**

static String getBin8**(**String str**)** **{**

int num **=** Integer**.**parseInt**(**str**);**

String bin **=** Integer**.**toBinaryString**(**num**);**

StringBuilder sb **=** **new** StringBuilder**(**bin**);**

int len **=** bin**.**length**();**

**if** **(**len **<** 8**)** **{**

int i **=** 8 **-** len**;**

**while** **(**i**--** **>** 0**)** **{**

sb**.**insert**(**0**,** '0'**);**

**}**

**}**

**return** sb**.**toString**();**

**}**

**}**

# 密码验证合格程序

|  |  |
| --- | --- |
| 描述 | 密码要求:        1.长度超过8位        2.包括大小写字母.数字.其它符号,以上四种至少三种        3.不能有相同长度超2的子串重复        说明:长度超过2的子串 |
| 知识点 | 字符串,数组 |
| 运行时间限制 | 0M |
| 内存限制 | 0 |
| 输入 | 一组或多组长度超过2的子符串。每组占一行 |
| 输出 | 如果符合要求输出：OK，否则输出NG    每行输出对应一组输入的结果； |
| 样例输入 | 021Abc9000 021Abc9Abc1 021ABC9000 021$bc9000 |
| 样例输出 | OK NG NG OK |

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

StringBuilder sb **=** **new** StringBuilder**();**

**while** **(**scanner**.**hasNextLine**())** **{**

String in **=** scanner**.**nextLine**();**

sb**.**append**(**check**(**in**)).**append**(**"\n"**);**

// System.out.println(sb.toString());

**}**

System**.**out**.**println**(**sb**.**toString**().**trim**());**

scanner**.**close**();**

**}**

static String check**(**String pwd**)** **{**

boolean res **=** **true;**

**if** **(**pwd**.**length**()** **<=** 8**)** **{**

res **=** **false;**

**}**

int**[]** flags **=** **new** int**[**4**];**

char ch**;**

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

ch **=** pwd**.**charAt**(**i**);**

**if** **(**ch **>=** 'a' **&&** ch **<=** 'z'**)** **{**

flags**[**0**]** **=** 1**;**

**}** **else** **if** **(**ch **>=** 'A' **&&** ch **<=** 'Z'**)** **{**

flags**[**1**]** **=** 1**;**

**}** **else** **if** **(**ch **>=** '0' **&&** ch **<=** '9'**)** **{**

flags**[**2**]** **=** 1**;**

**}** **else** **{**

flags**[**3**]** **=** 1**;**

**}**

**}**

**if** **(**flags**[**0**]+**flags**[**1**]+**flags**[**2**]+**flags**[**3**]** **<** 3**)** **{**

res **=** **false;**

**}**

**if** **(**isRepeat**(**pwd**))** **{**

res **=** **false;**

**}**

**if** **(**res**)**

**return** "OK"**;**

**return** "NG"**;**

**}**

static boolean isRepeat**(**String str**)** **{**

int id**=**0**;**

**while** **(**id **+** 3 **<=** str**.**length**())** **{**

String part **=** str**.**substring**(**id**,** id**+**3**);**

// System.out.println(part);

//不包含重复子串

**if** **(**str**.**substring**(**0**,** id**).**contains**(**part**)**

**||** str**.**substring**(**id**+**3**,** str**.**length**()).**contains**(**part**))** **{**

**return** **true;**

**}**

// System.out.println(str.substring(0, id));

// System.out.println(str.substring(id+3, str.length()));

id**++;**

**}**

**return** **false;**

**}**

**}**

# 简单错误记录

|  |  |
| --- | --- |
| 描述 | 开发一个简单错误记录功能小模块，能够记录出错的代码所在的文件名称和行号。    处理：    1、 记录最多8条错误记录，循环记录，对相同的错误记录（净文件名称和行号完全匹配）只记录一条，错误计数增加；    2、 超过16个字符的文件名称，只记录文件的最后有效16个字符；    3、 输入的文件可能带路径，记录文件名称不能带路径。 |
| 知识点 | 字符串 |
| 运行时间限制 | 0M |
| 内存限制 | 0 |
| 输入 | 一行或多行字符串。每行包括带路径文件名称，行号，以空格隔开。    如：E:\V1R2\product\fpgadrive.c   1325 |
| 输出 | 将所有的记录统计并将结果输出，格式：文件名 代码行数 数目，一个空格隔开，如：     fpgadrive.c 1325 1 |
| 样例输入 | E:\V1R2\product\fpgadrive.c 1325 |
| 样例输出 | fpgadrive.c 1325 1 |

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**HashMap**;**

**import** java**.**util**.**Scanner**;**

//E:\V1R2\product\fpgadrive.c 1325

public class Main **{**

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

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

HashMap**<**String**,** Log**>** map **=** **new** HashMap**<>();**

ArrayList**<**Log**>** list **=** **new** ArrayList**<>();**

**while** **(**scanner**.**hasNext**())** **{**

String fName **=** scanner**.**next**();**

**if** **(**fName**.**equals**(**"0"**))** **break;**

int line **=** scanner**.**nextInt**();**

Log log **=** **new** Log**(**fName**,** line**);**

int id **=** getId**(**list**,** log**);**

**if** **(**id **==** **-**1**)** **{**

list**.**add**(**log**);**

**}** **else** **{**

list**.**get**(**id**).**add**();**

**}**

**}**

//Collections.sort(list);

int len **=** list**.**size**();**

int s **=** len**>**8**?**len**-**8**:**0**;**

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

System**.**out**.**println**(**list**.**get**(**i**));**

**}**

scanner**.**close**();**

**}**

static int getId**(**ArrayList**<**Log**>** list**,** Log log**)** **{**

int ans **=** **-**1**;**

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

**if** **(**list**.**get**(**i**).**equals**(**log**))** **{**

ans **=** i**;**

**break;**

**}**

**}**

**return** ans**;**

**}**

**}**

class Log **implements** Comparable**<**Log**>{**

String fullName**;**

String name**;**

int line**;**

int num**;**

int times**;**

public Log**(**String fn**,** int line**)** **{**

fullName **=** fn**;**

name **=** getFileName**(**fullName**);**

**this.**line **=** line**;**

times**++;**

**}**

public void add**()** **{**

times**++;**

**}**

private String getFileName**(**String text**)** **{**

int id **=** text**.**length**()-**1**;**

StringBuilder sb **=** **new** StringBuilder**();**

int i**=**0**;**

**while** **(**i**++** **<** 16**)** **{**

char ch **=** text**.**charAt**(**id**);**

**if** **(**ch **==** '\\'**)** **break;**

sb**.**append**(**ch**);**

id**--;**

**}**

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

**}**

@Override

public boolean equals**(**Object obj**)** **{**

**if** **(**obj**.**getClass**()** **!=** Log**.**class**)** **return** **false;**

Log log **=** **(**Log**)** obj**;**

**return** log**.**name**.**equals**(**name**)** **&&** log**.**line **==** line**;**

**}**

@Override

public String toString**()** **{**

**return** name**+**" "**+**line**+**" "**+**times**;**

**}**

@Override

public int compareTo**(**Log o**)** **{**

int dif **=** o**.**times **-** **this.**times**;**

**return** dif**;**

**}**

**}**

没有通过OJ：

**import** java**.**util**.**LinkedList**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

LinkedList**<**Log**>** list **=** **new** LinkedList**<>();**

**while** **(**scanner**.**hasNextLine**())** **{**

String fullName **=** scanner**.**next**();**

int line **=** scanner**.**nextInt**();**

Log log **=** **new** Log**(**getFileName**(**fullName**),** line**);**

**if** **(**list**.**contains**(**log**))** **{**

int i **=** list**.**indexOf**(**log**);**

list**.**get**(**i**).**addCount**();**

**}** **else** **{**

list**.**add**(**log**);**

**}**

**if** **(**list**.**size**()** **>** 8**)** **{**

list**.**removeFirst**();**

**};**

// **if** **(**list**.**size**()** **>** 2**)** **break;**

// System.out.println(list);

**}**

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

System**.**out**.**println**(**list**.**get**(**i**));**

**}**

scanner**.**close**();**

**}**

static String getFileName**(**String fullName**)** **{**

StringBuilder sb **=** **new** StringBuilder**();**

int id **=** fullName**.**length**()-**1**;**

**while** **(**id **>=** 0**)** **{**

char ch **=** fullName**.**charAt**(**id**);**

**if** **(**ch **==** '\\'**)** **{**

**break;**

**}**

sb**.**append**(**ch**);**

**if** **(**sb**.**length**()** **==** 16**)** **break;**

id**--;**

**}**

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

**}**

**}**

class Log **{**

public String fileName**;**

public int line**;**

private int count**;**

public Log**(**String f**,** int l**)** **{**

fileName **=** f**;**

line **=** l**;**

count **=** 1**;**

**}**

public void addCount**()** **{**

count**++;**

**}**

@Override

public boolean equals**(**Object obj**)** **{**

**if** **(**obj **==** **null)** **{**

**return** **false;**

**}**

**if** **(**getClass**()** **!=** obj**.**getClass**())** **{**

**return** **false;**

**}**

Log log **=** **(**Log**)** obj**;**

**if** **(!**fileName**.**equals**(**log**.**fileName**)** **||** line **!=** log**.**line**)** **{**

**return** **false;**

**}**

**return** **true;**

**}**

@Override

public String toString**()** **{**

**return** fileName**+**" "**+**line**+**" "**+**count**;**

**}**

**}**

# 删除字符串中出现次数最少的字符

|  |  |
| --- | --- |
| 描述 | 实现删除字符串中出现次数最少的字符，若多个字符出现次数一样，则都删除。输出删除这些单词后的字符串，字符串中其它字符保持原来的顺序。 |
| 知识点 | 字符串 |
| 运行时间限制 | 0M |
| 内存限制 | 0 |
| 输入 | 字符串只包含小写英文字母, 不考虑非法输入，输入的字符串长度小于等于20个字节。 |
| 输出 | 删除字符串中出现次数最少的字符后的字符串。 |
| 样例输入 | abcdd |
| 样例输出 | dd |

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Collections**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String in **=** scanner**.**nextLine**();**

ArrayList**<**Log**>** list **=** **new** ArrayList**<>();**

char ch**;**

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

ch **=** in**.**charAt**(**i**);**

**if** **(**list **!=** **null)** **{**

int id **=** getId**(**list**,** ch**);**

**if** **(**id **==** **-**1**)** **{**

Log log **=** **new** Log**(**ch**);**

list**.**add**(**log**);**

**}** **else** **{**

list**.**get**(**id**).**add**();**

**}**

**}** **else** **{**

Log log **=** **new** Log**(**ch**);**

list**.**add**(**log**);**

**}**

**}**

Collections**.**sort**(**list**);**

int min **=** list**.**get**(**0**).**times**;**

StringBuilder sb **=** **new** StringBuilder**();**

sb**.**append**(**list**.**get**(**0**).**name**);**

int id **=** 1**;**

**while** **(**id **<** list**.**size**())** **{**

**if** **(**list**.**get**(**id**).**times **>** min**)** **{**

**break;**

**}** **else** **{**

sb**.**append**(**list**.**get**(**id**).**name**);**

**}**

id**++;**

**}**

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

String t **=** sb**.**charAt**(**i**)+**""**;**

in **=** in**.**replaceAll**(**t**,** ""**);**

**}**

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

**}**

scanner**.**close**();**

**}**

static int getId**(**ArrayList**<**Log**>** list**,** char ch**)** **{**

int id **=** **-**1**;**

int i**=**0**;**

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

**if** **(**list**.**get**(**i**).**name **==** ch**)**

**return** i**;**

**}**

**if** **(**i **==** list**.**size**())** **{**

**return** **-**1**;**

**}**

**return** **-**1**;**

**}**

**}**

class Log **implements** Comparable**<**Log**>{**

public char name**;**

public int times**;**

public Log**(**char n**)** **{**

name **=** n**;**

times **=** 1**;**

**}**

public void add**()** **{**

times**++;**

**}**

@Override

public String toString**()** **{**

**return** name**+**" "**+**times**;**

**}**

@Override

public int compareTo**(**Log o**)** **{**

// TODO Auto-generated method stub

**return** times **-** o**.**times**;**

**}**

**}**

# 素数伴侣

|  |  |
| --- | --- |
| 描述 | 题目描述 若两个正整数的和为素数，则这两个正整数称之为“素数伴侣”，如2和5、6和13，它们能应用于通信加密。现在密码学会请你设计一个程序，从已有的N（N为偶数）个正整数中挑选出若干对组成“素数伴侣”，挑选方案多种多样，例如有4个正整数：2，5，6，13，如果将5和6分为一组中只能得到一组“素数伴侣”，而将2和5、6和13编组将得到两组“素数伴侣”，能组成“素数伴侣”最多的方案称为“最佳方案”，当然密码学会希望你寻找出“最佳方案”。  输入:  有一个正偶数N（N≤100），表示待挑选的自然数的个数。后面给出具体的数字，范围为[2,30000]。  输出:  输出一个整数K，表示你求得的“最佳方案”组成“素数伴侣”的对数。 |
| 知识点 | 查找,搜索,排序 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入说明 1 输入一个正偶数n 2 输入n个整数 |
| 输出 | 求得的“最佳方案”组成“素数伴侣”的对数。 |
| 样例输入 | 4 2 5 6 13 |
| 样例输出 | 2 |

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

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

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

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

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

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

origin**[**i**]** **=** scanner**.**nextInt**();**

**}**

dfs**(**origin**,** get**,** book**,** 0**);**

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

**}**

scanner**.**close**();**

**}**

static int max **=** 0**;**

static void dfs**(**int**[]** origin**,** int**[]** get**,**int**[]** book**,** int n**)** **{**

**if** **(**n **==** origin**.**length**)** **{**

// print(get);

// System.out.println(getNum(get));

int num **=** getNum**(**get**);**

**if** **(**max **<** num**)** **{**

max **=** num**;**

**}**

**return;**

**}**

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

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

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

book**[**i**]** **=** 1**;**

get**[**n**]** **=** t**;**

dfs**(**origin**,** get**,** book**,** n**+**1**);**

book**[**i**]** **=** 0**;**

get**[**n**]** **=** 0**;**

**}**

**}**

**}**

static void print**(**int**[]** arr**)** **{**

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

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

**}**

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

**}**

static int getNum**(**int**[]** arr**)** **{**

int len **=** arr**.**length**;**

int num **=** 0**;**

**for** **(**int i**=**0**;** i**<**len**-**1**;** i**+=**2**)** **{**

**if** **(**isPrime**(**arr**[**i**]+**arr**[**i**+**1**]))** **{**

num**++;**

**}**

**}**

**return** num**;**

**}**

static boolean isPrime**(**int n**)** **{**

**if** **(**n **<** 2**)return** **false;**

int end **=** **(**int**)** Math**.**sqrt**(**n**);**

int i**=**2**;**

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

**if** **(**n **%** i **==** 0**)**

**return** **false;**

**}**

**return** **true;**

**}**

**}**

# 查找兄弟单词

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Collections**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

ArrayList**<**String**>** list **=** **new** ArrayList**<>();**

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

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

String s **=** scanner**.**next**();**

list**.**add**(**s**);**

**}**

String find **=** scanner**.**next**();**

int site **=** scanner**.**nextInt**();**

ArrayList**<**String**>** brothers **=** **new** ArrayList**<>();**

String get**;**

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

get **=** list**.**get**(**i**);**

**if** **(**get**.**equals**(**find**))** **{**

**continue;**

**}**

**if** **(**isBrother**(**find**,** get**))** **{**

brothers**.**add**(**get**);**

**}**

**}**

Collections**.**sort**(**brothers**);**

System**.**out**.**println**(**brothers**.**size**());**

**if** **(**brothers**.**size**()** **>=** site**)**

System**.**out**.**println**(**brothers**.**get**(**site**-**1**));**

**}**

scanner**.**close**();**

**}**

static boolean isBrother**(**String s1**,** String s2**)** **{**

**if** **(**s1**.**length**()** **!=** s2**.**length**())** **{**

**return** **false;**

**}**

**return** isEqual**(**getArr**(**s1**),** getArr**(**s2**));**

**}**

static int**[]** getArr**(**String text**)** **{**

int**[]** table **=** **new** int**[**26**];**

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

table**[**text**.**charAt**(**i**)-**'a'**]++;**

**}**

**return** table**;**

**}**

static boolean isEqual**(**int**[]** a**,** int**[]** b**)** **{**

**for** **(**int i**=**0**;** i**<**26**;** i**++)** **{**

**if** **(**a**[**i**]** **!=** b**[**i**])**

**return** **false;**

**}**

**return** **true;**

**}**

**}**

# 整数与IP地址间的转换

|  |  |
| --- | --- |
| 描述 | 原理：ip地址的每段可以看成是一个0-255的整数，把每段拆分成一个二进制形式组合起来，然后把这个二进制数转变成 一个长整数。 举例：一个ip地址为10.0.3.193 每段数字             相对应的二进制数 10                   00001010 0                    00000000 3                    00000011 193                  11000001 组合起来即为：00001010 00000000 00000011 11000001,转换为10进制数就是：167773121，即该IP地址转换后的数字就是它了。    的每段可以看成是一个0-255的整数，需要对IP地址进行校验 |
| 知识点 | 字符串,位运算 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入  1 输入IP地址 2 输入10进制型的IP地址 |
| 输出 | 输出 1 输出转换成10进制的IP地址 2 输出转换后的IP地址 |
| 样例输入 | 10.0.3.193 167969729 |
| 样例输出 | 167773121 10.3.3.193 |

package com**.**oj**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String ip **=** scanner**.**next**();**

String**[]** ips **=** ip**.**split**(**"\\."**);**

StringBuilder sb **=** **new** StringBuilder**();**

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

sb**.**append**(**get8Bin**(**ips**[**i**]));**

**}**

long ipNum **=** Long**.**parseLong**(**sb**.**toString**(),** 2**);**

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

long num **=** scanner**.**nextLong**();**

String ipStr **=** get32Bin**(**num**);**

int id**=**0**;**

**for** **(**int i**=**0**;** i**<**3**;** i**++)** **{**

System**.**out**.**print**(**getInt**(**ipStr**.**substring**(**id**,** id**+**8**))+**"."**);**

id **+=** 8**;**

**}**

System**.**out**.**print**(**getInt**(**ipStr**.**substring**(**id**)));**

**}**

scanner**.**close**();**

**}**

static String get8Bin**(**String str**)** **{**

int num **=** Integer**.**parseInt**(**str**);**

String bin **=** Integer**.**toBinaryString**(**num**);**

StringBuilder sb **=** **new** StringBuilder**(**bin**);**

**if** **(**bin**.**length**()** **<** 8**)** **{**

int id **=** 8 **-** bin**.**length**();**

**while** **(**id **>** 0**)** **{**

sb**.**insert**(**0**,** '0'**);**

id**--;**

**}**

**}**

**return** sb**.**toString**();**

**}**

static String get32Bin**(**long num**)** **{**

String bin **=** Long**.**toBinaryString**(**num**);**

StringBuilder sb **=** **new** StringBuilder**(**bin**);**

**if** **(**sb**.**length**()** **<=** 32**)** **{**

int id **=** 32 **-** bin**.**length**();**

**while** **(**id **>** 0**)** **{**

sb**.**insert**(**0**,** '0'**);**

id**--;**

**}**

bin **=** sb**.**toString**();**

**}** **else** **{**

bin **=** sb**.**substring**(**sb**.**length**()-**32**,** sb**.**length**());**

**}**

**return** bin**;**

**}**

static int getInt**(**String bin**)** **{**

**return** Integer**.**parseInt**(**bin**,** 2**);**

**}**

**}**

# 字符串运用-密码截取

|  |  |
| --- | --- |
| 描述 | Catcher是MCA国的情报员，他工作时发现敌国会用一些对称的密码进行通信，比如像这些ABBA，ABA，A，123321，但是他们有时会在开始或结束时加入一些无关的字符以防止别国破解。比如进行下列变化 ABBA->12ABBA,ABA->ABAKK,123321->51233214　。因为截获的串太长了，而且存在多种可能的情况（abaaab可看作是aba,或baaab的加密形式），Cathcer的工作量实在是太大了，他只能向电脑高手求助，你能帮Catcher找出最长的有效密码串吗？  样例输入:  ABBA  12ABBA  A  ABAKK  51233214  abaaab  样例输出:  4  4  1  3  6  5  可以使用中提供的库函数。  实现接口，每个接口实现1个基本操作：  voidGetCipherMaxLen(characCipherContent[],int \*piCipherLen)：  acCipherContent是一个字符串数组常量，见参考用例；  piCipherLen为输出有效密码串的最大长度；  题目框架中有2个参考用例，其它用例请执行编写。 |
| 知识点 | 字符串 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入一串字符 |
| 输出 | 输出有效长度 |
| 样例输入 | ABBA |
| 样例输出 | 4 |

**也可以使用反转求最长公共子串**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String in **=** scanner**.**next**();**

// System.out.println(getPwdLen(in));

StringBuilder sb **=** **new** StringBuilder**(**in**);**

System**.**out**.**println**(**getMaxLen**(**in**,** sb**.**reverse**().**toString**()));**

**}**

scanner**.**close**();**

**}**

private static int getMaxLen**(**String s1**,** String s2**)** **{**

int len **=** s1**.**length**();**

int dp**[][]** **=** **new** int**[**len**+**1**][**len**+**1**];**

int max **=** 0**;**

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

**for** **(**int j**=**1**;** j**<=**len**;** j**++)** **{**

**if** **(**s1**.**charAt**(**i**-**1**)** **==** s2**.**charAt**(**j**-**1**))** **{**

dp**[**i**][**j**]** **=** dp**[**i**-**1**][**j**-**1**]** **+** 1**;**

**if** **(**max **<** dp**[**i**][**j**])** **{**

max **=** dp**[**i**][**j**];**

**}**

**}**

**}**

**}**

**return** max**;**

**}**

private static int getPwdLen**(**String str**)** **{**

int len **=** str**.**length**();**

int start **=** 0**;**

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

**if** **(**start**+**len **<=** str**.**length**())** **{**

**if** **(**isDuiChen**(**str**.**substring**(**start**,** start**+**len**)))** **{**

**return** **(**str**.**substring**(**start**,** start**+**len**).**length**());**

**}**

start**++;**

**}** **else** **{**

start **=** 0**;**

len**--;**

**}**

**}**

**}**

static boolean isDuiChen**(**String str**)** **{**

int h**=**0**,** t **=** str**.**length**()-**1**;**

**while** **(**h **<** t**)** **{**

**if** **(**str**.**charAt**(**h**)** **!=** str**.**charAt**(**t**))** **{**

**return** **false;**

**}**

h**++;**

t**--;**

**}**

**return** **true;**

**}**

**}**

# 数独（Sudoku）

|  |  |
| --- | --- |
| 描述 | 问题描述：数独（Sudoku）是一款大众喜爱的数字逻辑游戏。  玩家需要根据9X9盘面上的已知数字，推算出所有剩余空格的数字，  并且满足每一行、每一列、每一个粗线宫内的数字均含1-9，并且不重复。 输入： 包含已知数字的9X9盘面数组[空缺位以数字0表示] 输出： 完整的9X9盘面数组 |
| 知识点 | 查找,搜索,排序 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 包含已知数字的9X9盘面数组[空缺位以数字0表示] |
| 输出 | 完整的9X9盘面数组 |
| 样例输入 | 0 9 2 4 8 1 7 6 3 4 1 3 7 6 2 9 8 5 8 6 7 3 5 9 4  1 2 6 2 4 1 9 5 3 7 8 7 5 9 8 4 3 1 2 6 1 3 8 6 2 7 5 9  4 2 7 1 5 3 8 6 4 9 3 8 6 9 1 4 2 5 7 0 4 5 2 7 6 8 3 1 |
| 样例输出 | 5 9 2 4 8 1 7 6 3 4 1 3 7 6 2 9 8 5 8 6 7 3 5 9 4 1 2 6  2 4 1 9 5 3 7 8 7 5 9 8 4 3 1 2 6 1 3 8 6 2 7 5 9 4 2 7 1  5 3 8 6 4 9 3 8 6 9 1 4 2 5 7 9 4 5 2 7 6 8 3 1 |

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

ArrayList**<**Point**>** list **=** **new** ArrayList**<>();**

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

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

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

ns**[**i**][**j**]** **=** scanner**.**nextInt**();**

**if** **(**ns**[**i**][**j**]** **==** 0**)** **{**

Point point **=** **new** Point**(**i**,** j**);**

list**.**add**(**point**);**

**}**

**}**

**}**

// for (int i=0; i<list.size(); i++) {

// List<Integer> list2 = getPossibleValue(ns, list.get(i).x, list.get(i).y);

// System.out.println(list2);

// }

dfsOne**(**ns**,** list**,** 0**);**

**}**

scanner**.**close**();**

**}**

//判断后枚举,输出一种结果

static boolean findFlag **=** **false;**

static void dfsOne**(**int**[][]** arr**,** ArrayList**<**Point**>** list**,** int id**)** **{**

**if** **(**id **==** list**.**size**())** **{**

**if** **(**check**(**arr**))** **{**

print**(**arr**);**

findFlag **=** **true;**

**}**

**return;**

**}**

**if** **(**findFlag**)** **return;**

Point point **=** list**.**get**(**id**);**

List**<**Integer**>** pV **=** getPossibleValue**(**arr**,** point**.**x**,** point**.**y**);**

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

// System.out.print(point+" ");

// System.out.println(pV);

arr**[**point**.**x**][**point**.**y**]** **=** pV**.**get**(**i**);**

dfsOne**(**arr**,** list**,** id**+**1**);**

arr**[**point**.**x**][**point**.**y**]** **=** 0**;**

**}**

**}**

//判断后枚举,输出所有结果

static void dfsAll**(**int**[][]** arr**,** ArrayList**<**Point**>** list**,** int id**)** **{**

**if** **(**id **==** list**.**size**())** **{**

**if** **(**check**(**arr**))** **{**

print**(**arr**);**

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

**}**

**return;**

**}**

Point point **=** list**.**get**(**id**);**

List**<**Integer**>** pV **=** getPossibleValue**(**arr**,** point**.**x**,** point**.**y**);**

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

// System.out.print(point+" ");

// System.out.println(pV);

arr**[**point**.**x**][**point**.**y**]** **=** pV**.**get**(**i**);**

dfsAll**(**arr**,** list**,** id**+**1**);**

arr**[**point**.**x**][**point**.**y**]** **=** 0**;**

**}**

**}**

//暴力枚举

static void dfs**(**int**[][]** arr**,** ArrayList**<**Point**>** list**,** int id**)** **{**

**if** **(**id **==** list**.**size**())** **{**

**if** **(**check**(**arr**))** **{**

print**(**arr**);**

**}**

**return;**

**}**

**for** **(**int i**=**1**;** i**<=**9**;** i**++)** **{**

Point point **=** list**.**get**(**id**);**

arr**[**point**.**x**][**point**.**y**]** **=** i**;**

dfs**(**arr**,** list**,** id**+**1**);**

arr**[**point**.**x**][**point**.**y**]** **=** 0**;**

**}**

**}**

static void print**(**int**[][]** a**)** **{**

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

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

**if** **(**j **==** 8**)**

System**.**out**.**printf**(**"%d\n"**,** a**[**i**][**j**]);**

**else**

System**.**out**.**printf**(**"%d "**,** a**[**i**][**j**]);**

**}**

**}**

**}**

//检查是否满足数独

static boolean check**(**int**[][]** arr**)** **{**

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

**if** **(!**checkRow**(**arr**,** i**)** **||** **!**checkColumn**(**arr**,** i**))** **{**

**return** **false;**

**}**

**}**

**if** **(!**checkAllZone**(**arr**))** **{**

**return** **false;**

**}**

**return** **true;**

**}**

//检测第row行是否满足1~9

static boolean checkRow**(**int**[][]** arr**,** int row**)** **{**

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

int sum **=** 0**;**

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

book**[**arr**[**row**][**i**]]** **=** 1**;**

**}**

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

// System.out.print(book[i]+ " ");

sum **+=** book**[**i**];**

**}**

**if** **(**sum **==** 9**)** **return** **true;**

**return** **false;**

**}**

//检测第row行是否满足1~9

static boolean checkColumn**(**int**[][]** arr**,** int column**)** **{**

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

int sum **=** 0**;**

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

book**[**arr**[**i**][**column**]]** **=** 1**;**

**}**

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

// System.out.print(book[i]+ " ");

sum **+=** book**[**i**];**

**}**

**if** **(**sum **==** 9**)** **return** **true;**

**return** **false;**

**}**

//检测3\*3区域是否满足1~9

static boolean checkPartZone**(**int**[][]** arr**,** int x**,** int y**)** **{**

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

int sum **=** 0**;**

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

book**[**arr**[**x**+**i**/**3**][**y**+**i**%**3**]]** **=** 1**;**

**}**

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

// System.out.print(book[i]+ " ");

sum **+=** book**[**i**];**

**}**

**if** **(**sum **==** 9**)** **return** **true;**

**return** **false;**

**}**

//检测9\*9区域是否满足1~9

static boolean checkAllZone**(**int**[][]** arr**)** **{**

**for** **(**int i**=**0**;** i**<=**6**;** i**+=**3**)** **{**

**for** **(**int j**=**0**;** j**<=**6**;** j**+=**3**)** **{**

// System.out.println(i+" " +j);

**if** **(!**checkPartZone**(**arr**,** i**,** j**))** **{**

**return** **false;**

**}**

**}**

**}**

**return** **true;**

**}**

//获取某个位置可能取的数值

static List**<**Integer**>** getPossibleValue**(**int**[][]** arr**,** int x**,** int y**)** **{**

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

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

int sum **=** 0**;**

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

book**[**arr**[**i**][**y**]]** **=** 1**;**

book**[**arr**[**x**][**i**]]** **=** 1**;**

**}**

//获取坐标所在3\*3区域的起点坐标

int startX **=** x**/**3**\***3**;**

int startY **=** y**/**3**\***3**;**

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

book**[**arr**[**startX**+**i**/**3**][**startY**+**i**%**3**]]** **=** 1**;**

**}**

**for** **(**int i**=**1**;** i**<=**9**;** i**++)** **{**

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

list**.**add**(**i**);**

**}**

**}**

**return** list**;**

**}**

**}**

class Point **{**

public int x**,**y**;**

public Point**(**int x**,** int y**)** **{**

**this.**x **=** x**;**

**this.**y **=** y**;**

**}**

@Override

public String toString**()** **{**

**return** x **+** " " **+** y**;**

**}**

**}**

//从空数组生成数独

**static** **boolean** *findOneFlag* = **false**;

**static** **void** dfsGet(**int**[][] arr, **int** num) {

**if** (num == 81) {

*print*(arr);

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

*findOneFlag* = **true**;

**return**;

}

**if** (*findOneFlag*) **return**;

**int** x = num / 9;

**int** y = num % 9;

List<Integer> pV = *getPossibleValue*(arr, x, y);

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

arr[x][y] = pV.get(i);

*dfsGet*(arr, num+1);

arr[x][y] = 0;

}

}

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**LinkedList**;**

**import** java**.**util**.**Scanner**;**

/\*

0 9 2 4 8 1 7 6 3 4 1 3 7 6 2 9 8 5 8 6 7 3 5 9 4

1 2 6 2 4 1 9 5 3 7 8 7 5 9 8 4 3 1 2 6 1 3 8 6 2 7 5 9

4 2 7 1 5 3 8 6 4 9 3 8 6 9 1 4 2 5 7 0 4 5 2 7 6 8 3 1

\*/

public class Demo **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

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

int zeroNums **=** 0**;**

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

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

arr**[**i**][**j**]** **=** scanner**.**nextInt**();**

**if** **(**arr**[**i**][**j**]** **==** 0**)** zeroNums**++;**

**}**

**}**

// print(arr);

findFlag **=** **false;**

dfs**(**arr**,** 0**,** zeroNums**);**

**}**

scanner**.**close**();**

**}**

static boolean findFlag **=** **false;**

static void dfs**(**int**[][]** arr**,** int id**,** int zeros**)** **{**

**if** **(**findFlag**)** **return;**

**if** **(**zeros **==** 0**)** **{**

print**(**arr**);**

// System.out.println();

findFlag **=** **true;**

**return;**

**};**

**while** **(**arr**[**id**/**9**][**id**%**9**]** **!=** 0**)** **{**

id**++;**

**}**

ArrayList**<**Integer**>** list **=** getValues**(**arr**,** id**/**9**,** id**%**9**);**

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

int num **=** list**.**get**(**i**);**

arr**[**id**/**9**][**id**%**9**]** **=** num**;**

dfs**(**arr**,** id**+**1**,** zeros**-**1**);**

arr**[**id**/**9**][**id**%**9**]** **=** 0**;**

**}**

**}**

static ArrayList**<**Integer**>** getValues**(**int**[][]** arr**,** int x**,** int y**)** **{**

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

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

book**[**arr**[**x**][**i**]]** **=** 1**;**

book**[**arr**[**i**][**y**]]** **=** 1**;**

**}**

int sx **=** x**/**3**\***3**;**

int sy **=** y**/**3**\***3**;**

**for** **(**int i**=**0**;** i**<**3**;** i**++)** **{**

**for** **(**int j**=**0**;** j**<**3**;** j**++)** **{**

book**[**arr**[**sx**+**i**][**sy**+**j**]]** **=** 1**;**

**}**

book**[**arr**[**x**][**i**]]** **=** 1**;**

book**[**arr**[**i**][**y**]]** **=** 1**;**

**}**

ArrayList**<**Integer**>** list **=** **new** ArrayList**<>();**

**for** **(**int i**=**1**;** i**<=**9**;** i**++)** **{**

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

list**.**add**(**i**);**

**}**

**}**

// System.out.println(list);

**return** list**;**

**}**

static void print**(**int**[][]** a**)** **{**

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

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

**if** **(**j **==** 8**&&**i**<**8**)**

System**.**out**.**printf**(**"%d\n"**,** a**[**i**][**j**]);**

**else** **if** **(**j **==** 8**&&**i **==** 8**)**

System**.**out**.**printf**(**"%d"**,** a**[**i**][**j**]);**

**else**

System**.**out**.**printf**(**"%d "**,** a**[**i**][**j**]);**

**}**

**}**

**}**

**}**

# 迷宫问题

|  |  |
| --- | --- |
| 描述 | 定义一个二维数组N\*M（其中2<=N<=10;2<=M<=10），如5 × 5数组下所示：  int maze[5][5] = {          0, 1, 0, 0, 0,          0, 1, 0, 1, 0,          0, 0, 0, 0, 0,          0, 1, 1, 1, 0,          0, 0, 0, 1, 0,  };  它表示一个迷宫，其中的1表示墙壁，0表示可以走的路，只能横着走或竖着走，不能斜着走，要求编程序找出从左  上角到右下角的最短路线。入口点为[0,0],既第一空格是可以走的路。  Input  一个N × M的二维数组，表示一个迷宫。数据保证有唯一解,不考虑有多解的情况，即迷宫只有一条通道。  Output  左上角到右下角的最短路径，格式如样例所示。  Sample Input  0 1 0 0 0  0 1 0 1 0  0 0 0 0 0  0 1 1 1 0  0 0 0 1 0  Sample Output  (0, 0)  (1, 0)  (2, 0)  (2, 1)  (2, 2)  (2, 3)  (2, 4)  (3, 4)  (4, 4) |
| 知识点 | 查找,搜索,排序 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入两个整数，分别表示二位数组的行数，列数。再输入相应的数组，其中的1表示墙壁，0表示可以走的路。  数据保证有唯一解,不考虑有多解的情况，即迷宫只有一条通道。 |
| 输出 | 左上角到右下角的最短路径，格式如样例所示。 |
| 样例输入 | 5 5 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 0 |
| 样例输出 | (0,0) (1,0) (2,0) (2,1) (2,2) (2,3) (2,4) (3,4) (4,4) |

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

**import** org**.**omg**.**Messaging**.**SyncScopeHelper**;**

public class Main **{**

static int**[][]** maze1 **=** **{**

**{**0**,** 1**,** 0**,** 0**,** 0**},**

**{**0**,** 1**,** 0**,** 1**,** 0**},**

**{**0**,** 0**,** 0**,** 0**,** 0**},**

**{**0**,** 1**,** 1**,** 1**,** 0**},**

**{**0**,** 0**,** 0**,** 1**,** 0**}};**

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

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

**while** **(**scanner**.**hasNext**())** **{**

int m **=** scanner**.**nextInt**();**

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

int**[][]** maze **=** **new** int**[**m**][**n**];**

int**[][]** book **=** **new** int**[**m**][**n**];**

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

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

maze**[**i**][**j**]** **=** scanner**.**nextInt**();**

**}**

**}**

// print(maze);

dfs**(**maze**,** book**,** 0**,** 0**,** 1**);**

printPoints**(**points**);**

**}**

scanner**.**close**();**

**}**

static int**[][]** next **=** **{{**0**,**1**},{**0**,-**1**},{**1**,**0**},{-**1**,**0**}};**

static int minSteps **=** Integer**.**MAX\_VALUE**;**

static int**[][]** points**;**

static void dfs**(**int**[][]** map**,** int**[][]** book**,** int x**,** int y**,** int steps**)** **{**

**if** **(**x **==** map**.**length**-**1 **&&** y **==** map**[**0**].**length**-**1**)** **{**

book**[**x**][**y**]** **=** steps**;**

**if** **(**minSteps **>** steps**)** **{**

minSteps **=** steps**;**

int**[][]** ps **=** **new** int**[**minSteps**][**2**];**

//获取路径坐标

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

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

**if** **(**book**[**i**][**j**]** **!=** 0**)** **{**

ps**[**book**[**i**][**j**]-**1**][**0**]** **=** i**;**

ps**[**book**[**i**][**j**]-**1**][**1**]** **=** j**;**

**}**

**}**

**}**

points **=** ps**;**

// print(ps);

**}**

book**[**x**][**y**]** **=** 0**;**

**return;**

**}**

book**[**x**][**y**]** **=** steps**;**

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

int toX **=** x**+**next**[**i**][**0**];**

int toY **=** y**+**next**[**i**][**1**];**

**if** **(**toX**>=** 0**&&**toX**<**map**.**length**&&**toY**>=**0**&&**toY**<**map**[**0**].**length**)** **{**

**if** **(**map**[**toX**][**toY**]** **==** 0 **&&** book**[**toX**][**toY**]** **==** 0**)** **{**

dfs**(**map**,** book**,** toX**,** toY**,** steps**+**1**);**

**}**

**}**

**}**

book**[**x**][**y**]** **=** 0**;**

**}**

static void print**(**int**[][]** a**)** **{**

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

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

**if** **(**j **==** a**[**0**].**length**-**1**)**

System**.**out**.**printf**(**"%d\n"**,** a**[**i**][**j**]);**

**else**

System**.**out**.**printf**(**"%d "**,** a**[**i**][**j**]);**

**}**

**}**

**}**

static void printPoints**(**int**[][]** a**)** **{**

**if** **(**a **==** **null)** **return;**

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

System**.**out**.**printf**(**"(%d,%d)\n"**,**a**[**i**][**0**],**a**[**i**][**1**]);**

**}**

**}**

**}**

# 按字节截取字符串

|  |  |
| --- | --- |
| 描述 | 编写一个截取字符串的函数，输入为一个字符串和字节数，输出为按字节截取的字符串。但是要保证汉字不被截半个，如"我ABC"4，应该截为"我AB"，输入"我ABC汉DEF"6，应该输出为"我ABC"而不是"我ABC+汉的半个"。     * 接口说明   原型：public String cutString(String s, int length) |
| 知识点 | 字符串 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入待截取的字符串及长度 |
| 输出 | 截取后的字符串 |
| 样例输入 | 我ABC汉DEF 6 |
| 样例输出 | 我ABC |

package com**.**oj**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String str **=** scanner**.**next**();**

int len **=** scanner**.**nextInt**();**

StringBuilder sb **=** **new** StringBuilder**();**

char ch**;**

int getNum**=**0**;**

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

ch **=** str**.**charAt**(**i**);**

**if** **(**isChinese**(**ch**))** **{**

getNum **+=** 2**;**

**}** **else** **{**

getNum**++;**

**}**

**if** **(**getNum **<=** len**)** **{**

sb**.**append**(**ch**);**

**}** **else** **{**

**break;**

**}**

**}**

// System.out.println(str);

System**.**out**.**println**(**sb**.**toString**());**

**}**

scanner**.**close**();**

**}**

static boolean isChinese**(**char ch**)** **{**

Character**.**UnicodeScript sc **=** Character**.**UnicodeScript**.**of**(**ch**);**

**if** **(**sc **==** Character**.**UnicodeScript**.**HAN**)** **{**

**return** **true;**

**}**

**return** **false;**

**}**

**}**

# 线性插值

|  |  |
| --- | --- |
| 描述 | 信号测量的结果包括测量编号和测量值。存在信号测量结果丢弃及测量结果重复的情况。      1.测量编号不连续的情况，认为是测量结果丢弃。对应测量结果丢弃的情况，需要进行插值操作以更准确的评估信号。    采用简化的一阶插值方法,由丢失的测量结果两头的测量值算出两者中间的丢失值。    假设第M个测量结果的测量值为A，第N个测量结果的测量值为B。则需要进行(N-M-1)个测量结果的插值处理。进行一阶线性插值估计的第N+i个测量结果的测量值为A+( (B-A)/(N-M) )\*i  (注：N的编号比M大。)    例如：只有测量编号为4的测量结果和测量编号为7的测量结果，测量值分别为4和10          则需要补充测量编号为5和6的测量结果。           其中测量编号为5的测量值=4 + ((10-4)/(7-4))\*1 = 6           其中测量编号为6的测量值=4 + ((10-4)/(7-4))\*2 = 8          2.测量编号相同，则认为测量结果重复，需要对丢弃后来出现的测量结果。      请根据以上规则进行测量结果的整理。  详细描述：  接口说明  原型：  intCleanUpMeasureInfo(MEASURE\_INFO\_STRUCT\* pOriMeasureInfo,intnOriMINum,intnMaxMIRst, MEASURE\_INFO\_STRUCT\* pMeasureInfoRst);  输入参数：          MEASURE\_INFO\_STRUCT\* pOriMeasureInfo:         原始测量结果内容，以结构数组方式存放。测量编号已经按升序排列。MEASURE\_INFO\_STRUCT定义包含编号和测量值，见OJ.h            int nOriMINum:                                原始测量结果个数。            int nMaxMIRst:                                整理的测量结果最大个数。  输入参数：      MEASURE\_INFO\_STRUCT\* pMeasureInfoRst：         整理的测量结果  返回值：      Int           整理的测量结果个数 |
| 知识点 | 数组,函数,指针,位运算,结构体 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入说明  1 输入两个整数m, n 2 输入m个数据组 |
| 输出 | 输出整理后的结果 |
| 样例输入 | 2 3 4 5 5 7 |
| 样例输出 | 4 5 5 7 |

package com**.**oj**;**

**import** java**.**util**.**LinkedList**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

int len **=** scanner**.**nextInt**();**

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

LinkedList**<**CalRes**>** list **=** **new** LinkedList**<>();**

getRecord**(**scanner**,** len**,** list**);**

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

System**.**out**.**println**(**list**.**get**(**i**));**

**}**

**}**

scanner**.**close**();**

**}**

private static void getRecord**(**Scanner scanner**,** int len**,** LinkedList**<**CalRes**>** list**)** **{**

int num**,** value**,** preNum**=**0**,** preValue**=**0**;**

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

num **=** scanner**.**nextInt**();**

value **=** scanner**.**nextInt**();**

**if** **(**list**.**size**()** **==** 0**)** **{**

list**.**add**(new** CalRes**(**num**,** value**));**

**}** **else** **{**

preNum **=** list**.**getLast**().**num**;**

preValue **=** list**.**getLast**().**value**;**

**if** **(**num **==** preNum **+** 1**)** **{**

list**.**add**(new** CalRes**(**num**,** value**));**

**}** **else** **if** **(**num **==** preNum**)** **{**

**continue;**

**}**

**else** **if** **(**num **>** preNum **+** 1**)** **{**

int k **=** num **-** preNum **-** 1**;**//插入k个值

**for** **(**int ins**=**1**;** ins**<=**k**;** ins**++)** **{**

list**.**add**(new** CalRes**(**preNum**+**ins

**,** preValue**+(**value**-**preValue**)/(**num**-**preNum**)\***ins**));**

**}**

list**.**add**(new** CalRes**(**num**,** value**));**

**}**

**}**

**}**

**}**

**}**

class CalRes **{**

int num**;**

int value**;**

public CalRes**(**int n**,** int v**)** **{**

num **=** n**;**

value **=** v**;**

**}**

@Override

public String toString**()** **{**

**return** num**+**" "**+**value**;**

**}**

**}**

# 从单向链表中删除指定值的节点

|  |  |
| --- | --- |
| 描述 | 输入一个单向链表和一个节点的值，从单向链表中删除等于该值的节点，删除后如果链表中无节点则返回空指针。  链表结点定义如下：  struct ListNode  {        int       m\_nKey;        ListNode\* m\_pNext;  };  详细描述：  本题为考察链表的插入和删除知识。  链表的值不能重复  构造过程，例如  1 -> 2  3 -> 2  5 -> 1  4 -> 5  7 -> 2  最后的链表的顺序为 2 7 3 1 5 4  删除 结点 2  则结果为 7 3 1 5 4 |
| 知识点 | 链表,指针,结构体 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 1 输入链表结点个数 2 输入头结点的值 3 按照格式插入各个结点 4 输入要删除的结点的值 |
| 输出 | 输出删除结点后的序列 |
| 样例输入 | 5 2 3 2 4 3 5 2 1 4 3 |
| 样例输出 | 2 1 5 4 |

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

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

int head **=** scanner**.**nextInt**();**

MyLink link **=** **new** MyLink**(**head**);**

**for** **(**int i**=**0**;** i**<**n**-**1**;** i**++)** **{**

int v1 **=** scanner**.**nextInt**();**

int v2 **=** scanner**.**nextInt**();**

link**.**append**(**v2**,** v1**);**

**}**

int del **=** scanner**.**nextInt**();**

link**.**delete**(**del**);**

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

**}**

scanner**.**close**();**

**}**

**}**

class MyLink **{**

class Node **{**

public int value**;**

public Node next**;**

public Node**(**int v**)** **{**

value **=** v**;**

next **=** **null;**

**}**

**}**

Node head**;**

public MyLink**(**int h**)** **{**

**this.**head **=** **new** Node**(**h**);**

**}**

//v2 插入到v1 之后

public void append**(**int v1**,** int v2**)** **{**

Node it **=** head**;**

**while** **(**it **!=** **null)** **{**

int temp **=** it**.**value**;**

**if** **(**temp **==** v1**)** **{**

Node n **=** **new** Node**(**v2**);**

n**.**next **=** it**.**next**;**

it**.**next **=** n**;**

**}**

it **=** it**.**next**;**

**}**

**}**

//删除值为obj的节点

public void delete**(**int obj**)** **{**

Node pre **=** **null;**

Node it **=** head**;**

**while** **(**it **!=** **null)** **{**

**if** **(**it**.**value **==** obj**)** **{**

**if** **(**pre **==** **null)** **{**

head **=** head**.**next**;**

**}** **else** **{**

pre**.**next **=** it**.**next**;**

**}**

**}**

pre **=** it**;**//保存要删除节点的上一个节点

it **=** it**.**next**;**

**}**

**}**

@Override

public String toString**()** **{**

Node it **=** head**;**

StringBuilder sb **=** **new** StringBuilder**();**

**while** **(**it **!=** **null)** **{**

sb**.**append**(**it**.**value**+**" "**);**

it **=** it**.**next**;**

**}**

**return** sb**.**toString**().**trim**();**

**}**

**}**

# 四则运算

|  |  |
| --- | --- |
| 描述 | 请实现如下接口      /\* 功能：四则运算       \* 输入：strExpression：字符串格式的算术表达式，如: "3+2\*{1+2\*[-4/(8-6)+7]}"           \* 返回：算术表达式的计算结果       \*/  **public** **static** **int** calculate(String strExpression)      {          /\* 请实现\*/  **return** 0;      }  约束：   1. pucExpression字符串中的有效字符包括[‘0’-‘9’],‘+’,‘-’, ‘\*’,‘/’ ,‘(’， ‘)’,‘[’, ‘]’,‘{’ ,‘}’。 2. pucExpression算术表达式的有效性由调用者保证; |
| 知识点 | 栈 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入一个算术表达式 |
| 输出 | 得到计算结果 |
| 样例输入 | 3+2\*{1+2\*[-4/(8-6)+7]} |
| 样例输出 | 25 |

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

//9+(3-1)\*3+10/2

//[9, 3, 1, -, 3, \*, +, 10, 2, /, +]

public class Main **{**

static String symbolRegex **=** "[\*\\+-/\\]\\[)(}{]"**;**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String in **=** scanner**.**next**();**

//将{}、[]转换为（）

in **=** in**.**replaceAll**(**"[{\\[]"**,** "("**);**

in **=** in**.**replaceAll**(**"[}\\]]"**,** ")"**);**

// System.out.println(parseToList(in));

// System.out.println(getExpression(parseToList(in)));

System**.**out**.**println**(**cal**(**getExpression**(**parseToList**(**in**))));**

**}**

scanner**.**close**();**

**}**

//将输入表达式拆分，数字和符号依次存放

private static List**<**String**>** parseToList**(**String in**)** **{**

ArrayList**<**String**>** list **=** **new** ArrayList**<>();**

char ch**;**

int i**=**0**;**

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

ch **=** in**.**charAt**(**i**);**

**if** **(**Character**.**isDigit**(**ch**)** **||** //负数判断

**(**ch **==** '-'**&&(**i**-**1**>=**0**)&&!**Character**.**isDigit**(**in**.**charAt**(**i**-**1**)))**

**||** **(**i**==**0**&&**ch**==**'-'**))** **{**

String t **=** ch**+**""**;**

**if** **(++**i **==** in**.**length**())** **{**

list**.**add**(**t**);**

**break;**

**}**

ch **=** in**.**charAt**(**i**);**

**while** **(**Character**.**isDigit**(**ch**))** **{**

t **=** t **+** ch**;**

**if** **(++**i **==** in**.**length**())** **break;**

ch **=** in**.**charAt**(**i**);**

**}**

list**.**add**(**t**);**

**}** **else** **{**

list**.**add**(**ch**+**""**);**

i**++;**

**}**

**}**

**return** list**;**

**}**

//中缀表达式转换为后缀表达式

//数字直接输出，符号入栈，其中符号入栈前需要进行判断当前符号与栈顶符号的优先级

//只有当前符号的优先级高于栈顶符号才可以直接入栈。

//如果是（，则直接入栈，如果是），

//则将栈顶元素依次出栈，直到找到（为止。其中符号输出,(被忽略

//如果是\*或者/，因为它们的优先级最高，直接入栈

//如果是+或者-，如果栈顶为\* / + -，其优先级高于或等于入栈的+-

//则将其出栈，直至遇见（或者栈为空，再将+、-入栈

static List**<**String**>** getExpression**(**List**<**String**>** in**)** **{**

MyStack stack **=** **new** MyStack**();**

List**<**String**>** list **=** **new** ArrayList**<>();**

String ch**;**

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

ch **=** in**.**get**(**i**);**

**if** **(!**ch**.**matches**(**symbolRegex**))** **{**//数字

list**.**add**(**ch**);**

**}** **else** **{**//符号

**if** **(**ch**.**equals**(**")"**))** **{**

**do** **{**

String get **=** stack**.**pop**();**

**if** **(**get **==** **null** **||** get**.**equals**(**"("**))** **{**

**break;**

**}** **else** **{**

list**.**add**(**get**);**

**}**

**}** **while** **(true);**

**}**

**else** **if** **(**ch**.**equals**(**"+"**)** **||** ch**.**equals**(**"-"**))** **{**

String get **=** stack**.**pop**();**

**if** **(**get **==** **null)** **{**

stack**.**push**(**ch**);**

**}** **else** **if** **(**get**.**equals**(**"("**))** **{**

stack**.**push**(**get**);**

stack**.**push**(**ch**);**

**}** **else** **{**//为\*或/ + -

list**.**add**(**get**);**

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

get **=** stack**.**pop**();**

**if** **(**get **==** **null)** **{**

**break;**

**}** **else** **if** **(!**get**.**equals**(**"("**))** **{**

list**.**add**(**get**);**

**}** **else** **if** **(**get**.**equals**(**"("**)){**

stack**.**push**(**get**);**

**break;**

**}**

**}**

stack**.**push**(**ch**);**

**}**

**}**

**else** **{**//\* / (直接入栈

stack**.**push**(**ch**);**

**}**

**}**

//元素输出完毕，将栈内元素出栈

**if** **(**i **==** in**.**size**()** **-** 1**)** **{**

**do** **{**

String get **=** stack**.**pop**();**

**if** **(**get **==** **null)** **{**

**break;**

**}** **else** **{**

list**.**add**(**get**);**

**}**

**}** **while** **(true);**

**}**

**}**

**return** list**;**

**}**

//对后缀表达式运算求结果

//遇到数字直接入栈，遇到符号，出栈2个数字，对其进行相应的运算，再将结果入栈

//最后栈内的数据就是结果

//如符号-，依次出栈数字为d1,d2,计算d2-d1,对d2-d1入栈，即先出的数字在运算符右边

static int cal**(**List**<**String**>** list**)** **{**

MyStack stack **=** **new** MyStack**();**

String ch**;**

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

ch **=** list**.**get**(**i**);**

**if** **(!**ch**.**matches**(**symbolRegex**))** **{**

stack**.**push**(**ch**);**

**}** **else** **{**

int d2 **=** Integer**.**parseInt**(**String**.**valueOf**(**stack**.**pop**()));**

int d1 **=** Integer**.**parseInt**(**String**.**valueOf**(**stack**.**pop**()));**

**if** **(**ch**.**equals**(**"+"**))** **{**

int res **=** d1 **+** d2**;**

stack**.**push**(**res**+**""**);**

**}**

**if** **(**ch**.**equals**(**"-"**))** **{**

int res **=** d1 **-** d2**;**

stack**.**push**(**res**+**""**);**

**}**

**if** **(**ch**.**equals**(**"\*"**))** **{**

int res **=** d1 **\*** d2**;**

stack**.**push**(**res**+**""**);**

**}**

**if** **(**ch**.**equals**(**"/"**))** **{**

int res **=** d1 **/** d2**;**

stack**.**push**(**res**+**""**);**

**}**

**}**

**}**

**return** Integer**.**parseInt**(**stack**.**pop**());**

**}**

**}**

class MyStack **{**

String**[]** data **=** **new** String**[**1024**];**

int top **=** **-**1**;**

public void push**(**String ch**)** **{**

**if** **(**ch **==** **null)** **return;**

top**++;**

data**[**top**]** **=** ch**;**

**}**

public String pop**()** **{**

String ch **=** **null;**

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

ch **=** data**[**top**];**

top**--;**

**}**

**return** ch**;**

**}**

@Override

public String toString**()** **{**

StringBuilder sb **=** **new** StringBuilder**();**

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

sb**.**append**(**data**[**i**]+**" "**);**

**}**

**return** sb**.**toString**().**trim**();**

**}**

**}**

可以对小数进行四则运算

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

//9+(3-1)\*3+10/2

//[9, 3, 1, -, 3, \*, +, 10, 2, /, +]

public class Main **{**

static String symbolRegex **=** "[\*\\+-/\\]\\[)(}{]"**;**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String in **=** scanner**.**next**();**

//将{}、[]转换为（）

in **=** in**.**replaceAll**(**"[{\\[]"**,** "("**);**

in **=** in**.**replaceAll**(**"[}\\]]"**,** ")"**);**

System**.**out**.**println**(**parseToList**(**in**));**

// System.out.println(getExpression(parseToList(in)));

System**.**out**.**printf**(**"%.3f"**,** cal**(**getExpression**(**parseToList**(**in**))));**

**}**

scanner**.**close**();**

**}**

//将输入表达式拆分，数字和符号依次存放

private static List**<**String**>** parseToList**(**String in**)** **{**

ArrayList**<**String**>** list **=** **new** ArrayList**<>();**

char ch**;**

int i**=**0**;**

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

ch **=** in**.**charAt**(**i**);**

**if** **(**Character**.**isDigit**(**ch**)** **||** //负数判断

**(**ch **==** '-'**&&(**i**-**1**>=**0**)&&!**Character**.**isDigit**(**in**.**charAt**(**i**-**1**)))**

**||** **(**i**==**0**&&**ch**==**'-'**))** **{**

String t **=** ch**+**""**;**

**if** **(++**i **==** in**.**length**())** **{**

list**.**add**(**t**);**

**break;**

**}**

ch **=** in**.**charAt**(**i**);**

**while** **(**Character**.**isDigit**(**ch**)** **||** ch **==** '.'**)** **{**

t **=** t **+** ch**;**

**if** **(++**i **==** in**.**length**())** **break;**

ch **=** in**.**charAt**(**i**);**

**}**

list**.**add**(**t**);**

**}** **else** **{**

list**.**add**(**ch**+**""**);**

i**++;**

**}**

**}**

**return** list**;**

**}**

//中缀表达式转换为后缀表达式

//数字直接输出，符号入栈，其中符号入栈前需要进行判断当前符号与栈顶符号的优先级

//只有当前符号的优先级高于栈顶符号才可以直接入栈。

//如果是（，则直接入栈，如果是），

//则将栈顶元素依次出栈，直到找到（为止。其中符号输出,(被忽略

//如果是\*或者/，因为它们的优先级最高，直接入栈

//如果是+或者-，如果栈顶为\* / + -，其优先级高于或等于入栈的+-

//则将其出栈，直至遇见（或者栈为空，再将+、-入栈

static List**<**String**>** getExpression**(**List**<**String**>** in**)** **{**

MyStack stack **=** **new** MyStack**();**

List**<**String**>** list **=** **new** ArrayList**<>();**

String ch**;**

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

ch **=** in**.**get**(**i**);**

**if** **(!**ch**.**matches**(**symbolRegex**))** **{**//数字

list**.**add**(**ch**);**

**}** **else** **{**//符号

**if** **(**ch**.**equals**(**")"**))** **{**

**do** **{**

String get **=** stack**.**pop**();**

**if** **(**get **==** **null** **||** get**.**equals**(**"("**))** **{**

**break;**

**}** **else** **{**

list**.**add**(**get**);**

**}**

**}** **while** **(true);**

**}**

**else** **if** **(**ch**.**equals**(**"+"**)** **||** ch**.**equals**(**"-"**))** **{**

String get **=** stack**.**pop**();**

**if** **(**get **==** **null)** **{**

stack**.**push**(**ch**);**

**}** **else** **if** **(**get**.**equals**(**"("**))** **{**

stack**.**push**(**get**);**

stack**.**push**(**ch**);**

**}** **else** **{**//为\*或/ + -

list**.**add**(**get**);**

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

get **=** stack**.**pop**();**

**if** **(**get **==** **null)** **{**

**break;**

**}** **else** **if** **(!**get**.**equals**(**"("**))** **{**

list**.**add**(**get**);**

**}** **else** **if** **(**get**.**equals**(**"("**)){**

stack**.**push**(**get**);**

**break;**

**}**

**}**

stack**.**push**(**ch**);**

**}**

**}**

**else** **{**//\* / (直接入栈

stack**.**push**(**ch**);**

**}**

**}**

//元素输出完毕，将栈内元素出栈

**if** **(**i **==** in**.**size**()** **-** 1**)** **{**

**do** **{**

String get **=** stack**.**pop**();**

**if** **(**get **==** **null)** **{**

**break;**

**}** **else** **{**

list**.**add**(**get**);**

**}**

**}** **while** **(true);**

**}**

**}**

**return** list**;**

**}**

//对后缀表达式运算求结果

//遇到数字直接入栈，遇到符号，出栈2个数字，对其进行相应的运算，再将结果入栈

//最后栈内的数据就是结果

//如符号-，依次出栈数字为d1,d2,计算d2-d1,对d2-d1入栈，即先出的数字在运算符右边

static double cal**(**List**<**String**>** list**)** **{**

MyStack stack **=** **new** MyStack**();**

String ch**;**

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

ch **=** list**.**get**(**i**);**

**if** **(!**ch**.**matches**(**symbolRegex**))** **{**

stack**.**push**(**ch**);**

**}** **else** **{**

double d2 **=** Double**.**parseDouble**(**String**.**valueOf**(**stack**.**pop**()));**

double d1 **=** Double**.**parseDouble**(**String**.**valueOf**(**stack**.**pop**()));**

double res**;**

**if** **(**ch**.**equals**(**"+"**))** **{**

res **=** d1 **+** d2**;**

stack**.**push**(**res**+**""**);**

**}**

**if** **(**ch**.**equals**(**"-"**))** **{**

res **=** d1 **-** d2**;**

stack**.**push**(**res**+**""**);**

**}**

**if** **(**ch**.**equals**(**"\*"**))** **{**

res **=** d1 **\*** d2**;**

stack**.**push**(**res**+**""**);**

**}**

**if** **(**ch**.**equals**(**"/"**))** **{**

res **=** d1 **/** d2**;**

stack**.**push**(**res**+**""**);**

**}**

// int d2 = Integer.parseInt(String.valueOf(stack.pop()));

// int d1 = Integer.parseInt(String.valueOf(stack.pop()));

// if (ch.equals("+")) {

// int res = d1 + d2;

// stack.push(res+"");

// }

// if (ch.equals("-")) {

// int res = d1 - d2;

// stack.push(res+"");

// }

// if (ch.equals("\*")) {

// int res = d1 \* d2;

// stack.push(res+"");

// }

// if (ch.equals("/")) {

// int res = d1 / d2;

// stack.push(res+"");

// }

**}**

**}**

**return** Double**.**parseDouble**(**stack**.**pop**());**

**}**

**}**

class MyStack **{**

String**[]** data **=** **new** String**[**1024**];**

int top **=** **-**1**;**

public void push**(**String ch**)** **{**

**if** **(**ch **==** **null)** **return;**

top**++;**

data**[**top**]** **=** ch**;**

**}**

public String pop**()** **{**

String ch **=** **null;**

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

ch **=** data**[**top**];**

top**--;**

**}**

**return** ch**;**

**}**

@Override

public String toString**()** **{**

StringBuilder sb **=** **new** StringBuilder**();**

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

sb**.**append**(**data**[**i**]+**" "**);**

**}**

**return** sb**.**toString**().**trim**();**

**}**

**}**

可以识别输入错误

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

//9+(3-1)\*3+10/2

//[9, 3, 1, -, 3, \*, +, 10, 2, /, +]

public class Main **{**

static String symbolRegex **=** "[\*\\+-/\\]\\[)(}{]"**;**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String in **=** scanner**.**nextLine**();**

//将{}、[]转换为（）

in **=** in**.**replaceAll**(**"[{\\[]"**,** "("**);**

in **=** in**.**replaceAll**(**"[}\\]]"**,** ")"**);**

// System.out.println(parseToList(in));

// System.out.println(getExpression(parseToList(in)));

**try** **{**

int res **=** cal**(**getExpression**(**parseToList**(**in**)));**

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

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

**}** **catch** **(**Exception e**)** **{**

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

**}**

**}**

scanner**.**close**();**

**}**

//将输入表达式拆分，数字和符号依次存放

private static List**<**String**>** parseToList**(**String in**)** **{**

ArrayList**<**String**>** list **=** **new** ArrayList**<>();**

char ch**;**

int i**=**0**;**

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

ch **=** in**.**charAt**(**i**);**

**if** **(**Character**.**isDigit**(**ch**)** **||** //负数判断

**(**ch **==** '-'**&&(**i**-**1**>=**0**)&&!**Character**.**isDigit**(**in**.**charAt**(**i**-**1**)))**

**||** **(**i**==**0**&&**ch**==**'-'**))** **{**

String t **=** ch**+**""**;**

**if** **(++**i **==** in**.**length**())** **{**

list**.**add**(**t**);**

**break;**

**}**

ch **=** in**.**charAt**(**i**);**

**while** **(**Character**.**isDigit**(**ch**))** **{**

t **=** t **+** ch**;**

**if** **(++**i **==** in**.**length**())** **break;**

ch **=** in**.**charAt**(**i**);**

**}**

list**.**add**(**t**);**

**}** **else** **{**

list**.**add**(**ch**+**""**);**

i**++;**

**}**

**}**

**return** list**;**

**}**

//中缀表达式转换为后缀表达式

//数字直接输出，符号入栈，其中符号入栈前需要进行判断当前符号与栈顶符号的优先级

//只有当前符号的优先级高于栈顶符号才可以直接入栈。

//如果是（，则直接入栈，如果是），

//则将栈顶元素依次出栈，直到找到（为止。其中符号输出,(被忽略

//如果是\*或者/，因为它们的优先级最高，直接入栈

//如果是+或者-，如果栈顶为\* / + -，其优先级高于或等于入栈的+-

//则将其出栈，直至遇见（或者栈为空，再将+、-入栈

static List**<**String**>** getExpression**(**List**<**String**>** in**)** **{**

MyStack stack **=** **new** MyStack**();**

List**<**String**>** list **=** **new** ArrayList**<>();**

String ch**;**

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

ch **=** in**.**get**(**i**);**

**if** **(!**ch**.**matches**(**symbolRegex**))** **{**//数字

list**.**add**(**ch**);**

**}** **else** **{**//符号

**if** **(**ch**.**equals**(**")"**))** **{**

**do** **{**

String get **=** stack**.**pop**();**

**if** **(**get **==** **null** **||** get**.**equals**(**"("**))** **{**

**break;**

**}** **else** **{**

list**.**add**(**get**);**

**}**

**}** **while** **(true);**

**}**

**else** **if** **(**ch**.**equals**(**"+"**)** **||** ch**.**equals**(**"-"**))** **{**

String get **=** stack**.**pop**();**

**if** **(**get **==** **null)** **{**

stack**.**push**(**ch**);**

**}** **else** **if** **(**get**.**equals**(**"("**))** **{**

stack**.**push**(**get**);**

stack**.**push**(**ch**);**

**}** **else** **{**//为\*或/ + -

list**.**add**(**get**);**

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

get **=** stack**.**pop**();**

**if** **(**get **==** **null)** **{**

**break;**

**}** **else** **if** **(!**get**.**equals**(**"("**))** **{**

list**.**add**(**get**);**

**}** **else** **if** **(**get**.**equals**(**"("**)){**

stack**.**push**(**get**);**

**break;**

**}**

**}**

stack**.**push**(**ch**);**

**}**

**}**

**else** **{**//\* / (直接入栈

stack**.**push**(**ch**);**

**}**

**}**

//元素输出完毕，将栈内元素出栈

**if** **(**i **==** in**.**size**()** **-** 1**)** **{**

**do** **{**

String get **=** stack**.**pop**();**

**if** **(**get **==** **null)** **{**

**break;**

**}** **else** **{**

list**.**add**(**get**);**

**}**

**}** **while** **(true);**

**}**

**}**

**return** list**;**

**}**

//对后缀表达式运算求结果

//遇到数字直接入栈，遇到符号，出栈2个数字，对其进行相应的运算，再将结果入栈

//最后栈内的数据就是结果

//如符号-，依次出栈数字为d1,d2,计算d2-d1,对d2-d1入栈，即先出的数字在运算符右边

static int cal**(**List**<**String**>** list**)** **throws** Exception **{**

MyStack stack **=** **new** MyStack**();**

String ch**;**

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

ch **=** list**.**get**(**i**);**

**if** **(!**ch**.**matches**(**symbolRegex**))** **{**

stack**.**push**(**ch**);**

**}** **else** **{**

int d2 **=** Integer**.**parseInt**(**String**.**valueOf**(**stack**.**pop**()));**

int d1 **=** Integer**.**parseInt**(**String**.**valueOf**(**stack**.**pop**()));**

**if** **(**ch**.**equals**(**"+"**))** **{**

int res **=** d1 **+** d2**;**

stack**.**push**(**res**+**""**);**

**}**

**if** **(**ch**.**equals**(**"-"**))** **{**

int res **=** d1 **-** d2**;**

stack**.**push**(**res**+**""**);**

**}**

**if** **(**ch**.**equals**(**"\*"**))** **{**

int res **=** d1 **\*** d2**;**

stack**.**push**(**res**+**""**);**

**}**

**if** **(**ch**.**equals**(**"/"**))** **{**

int res **=** d1 **/** d2**;**

stack**.**push**(**res**+**""**);**

**}**

**}**

**}**

int r **=** Integer**.**parseInt**(**stack**.**pop**());**

//判断栈内是否只剩下一个元素，否则为无效计算

**if** **(**stack**.**pop**()** **==** **null)** **{**

**return** r**;**

**}** **else** **{**

**throw** **new** Exception**(**"input wrong"**);**

**}**

// return Integer.parseInt(stack.pop());

**}**

**}**

class MyStack **{**

String**[]** data **=** **new** String**[**1024**];**

int top **=** **-**1**;**

public void push**(**String ch**)** **{**

**if** **(**ch **==** **null)** **return;**

top**++;**

data**[**top**]** **=** ch**;**

**}**

public String pop**()** **{**

String ch **=** **null;**

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

ch **=** data**[**top**];**

top**--;**

**}**

**return** ch**;**

**}**

@Override

public String toString**()** **{**

StringBuilder sb **=** **new** StringBuilder**();**

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

sb**.**append**(**data**[**i**]+**" "**);**

**}**

**return** sb**.**toString**().**trim**();**

**}**

**}**

# MP3光标位置

|  |  |
| --- | --- |
| 描述 | MP3 Player因为屏幕较小，显示歌曲列表的时候每屏只能显示几首歌曲，用户要通过上下键才能浏览所有的歌曲。  为了简化处理，假设每屏只能显示4首歌曲，光标初始的位置为第1首歌。    现在要实现通过上下键控制光标移动来浏览歌曲列表，控制逻辑如下：   1. 歌曲总数<=4的时候，不需要翻页，只是挪动光标位置。光标在第一首歌曲上时，按Up键光标挪到最后一首歌曲； 2. 光标在最后一首歌曲时，按Down键光标挪到第一首歌曲。   其他情况下用户按Up键，光标挪到上一首歌曲；用户按Down键，光标挪到下一首歌曲。  歌曲总数大于4的时候（以一共有10首歌为例）：  特殊翻页：屏幕显示的是第一页（即显示第1 –4首）时，光标在第一首歌曲上，用户按Up键后，屏幕要显示最后一页  （即显示第7-10首歌），同时光标放到最后一首歌上。同样的，屏幕显示最后一页时，光标在最后一首歌曲上，  用户按Down键，屏幕要显示第一页，光标挪到第一首歌上。  一般翻页：屏幕显示的不是第一页时，光标在当前屏幕显示的第一首歌曲时，用户按Up键后，屏幕从当前歌曲  的上一首开始显示，光标也挪到上一首歌曲。光标当前屏幕的最后一首歌时的Down键处理也类似。  其他情况，不用翻页，只是挪动光标就行。    **详细描述：**   * 接口说明   原型：   * void InputPreCondition(int iTotalItemsNum,char\* pCommandLst, char\* pCurDspSongLst, int \* iCurItemNo);   输入参数：      int iTotalItemsNum：MP3中歌曲的总数      char\* pCommandLst：用户操作序列(‘U’表示Up键，‘D’表示Down键)  输出参数：      char\* pCurDspSongLst: 经过用户操作序列后，当前屏幕里显示的歌曲列表(从上到下)      int \* iCurItemNo：经过用户操作后光标的位置(第几首歌)  返回值：          void |
| 知识点 | 数组,函数 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入说明： 1 输入歌曲数量 2 输入命令 U或者D |
| 输出 | 输出说明 1 输出当前列表 2 输出当前选中歌曲 |
| 样例输入 | 10 UUUU |
| 样例输出 | 7 8 9 10 7 |

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

total **=** scanner**.**nextInt**();**

current **=** 1**;**

pageFirst **=** 1**;**

String cmds **=** scanner**.**next**();**

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

char cmd **=** cmds**.**charAt**(**i**);**

command**(**cmd**);**

**}**

showInfo**();**

**}**

scanner**.**close**();**

**}**

static int total**=**6**,** current**=**1**,**pageFirst**=**1**;**

static void command**(**char cmd**)** **{**

**if** **(**total **<=** 4**)** **{**

**if** **(**cmd **==** 'D'**)** **{**

current**++;**

**if** **(**current **==** total **+** 1**)** current **=** 1**;**

**}**

**if** **(**cmd **==** 'U'**)** **{**

current**--;**

**if** **(**current **==** 0**)** current **=** total**;**

**}**

**return;**

**}**

**if** **(**cmd **==** 'D'**)** **{**

current**++;**

**if** **(**current **==** total **+** 1**)** **{**

current **=** 1**;**

pageFirst **=** 1**;**

**}**

**if** **(**current **==** pageFirst**+**4**)** **{**

pageFirst**++;**

**}**

**}**

**if** **(**cmd **==** 'U'**)** **{**

current**--;**

**if** **(**current **==** 0**)** **{**

pageFirst **=** total**-**3**;**

current **=** total**;**

**}**

**if** **(**current **==** pageFirst**-**1**)** **{**

pageFirst**--;**

**}**

**}**

**}**

static void showInfo**()** **{**

**if** **(**total **<=** 4**)** **{**

**for** **(**int i**=**0**;** i**<**total**-**1**;** i**++)** **{**

System**.**out**.**printf**(**"%d "**,**pageFirst**+**i**);**

**}**

System**.**out**.**printf**(**"%d\n"**,**pageFirst**+**total**-**1**);**

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

**}** **else** **{**

**for** **(**int i**=**0**;** i**<**3**;** i**++)** **{**

System**.**out**.**printf**(**"%d "**,**pageFirst**+**i**);**

**}**

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

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

**}**

**}**

**}**

# 成绩排序

|  |  |
| --- | --- |
| 描述 | 查找和排序  题目：输入任意（用户，成绩）序列，可以获得成绩从高到低或从低到高的排列,相同成绩       都按先录入排列在前的规则处理。     例示：    jack      70    peter     96    Tom       70    smith     67     从高到低  成绩                peter     96        jack      70        Tom       70        smith     67     从低到高     smith     67  jack      70     Tom       70           peter     96 |
| 知识点 | 排序 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入多行，先输入要排序的人的个数，然后分别输入他们的名字和成绩，以一个空格隔开  用户名字，字符串,非空串,长度不超过20;  成绩，整数，范围【0，100】  输入排序方式，0或1，0表示成绩从高到低方式输出名字和成绩，1表示成绩从低到高方式输出名字和成绩 |
| 输出 | 按照指定方式输出名字和成绩，名字和成绩之间以一个空格隔开 |
| 样例输入 | 3 0 fang 90 yang 50 ning 70 |
| 样例输出 | fang 90 ning 70 yang 50 |

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Collections**;**

**import** java**.**util**.**Comparator**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

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

int type **=** scanner**.**nextInt**();**

ArrayList**<**Stu**>** list **=** **new** ArrayList**<>();**

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

list**.**add**(new** Stu**(**scanner**.**next**(),** i**+**1**,** scanner**.**nextInt**()));**

**}**

**if** **(**type **==** 1**)** **{**

Collections**.**sort**(**list**,** **new** Comparator**<**Stu**>()** **{**

@Override

public int compare**(**Stu o1**,** Stu o2**)** **{**

int dif **=** o1**.**score **-** o2**.**score**;**

**if** **(**dif **==** 0**)** **{**

dif **=** o1**.**num **-** o2**.**num**;**

**}**

**return** dif**;**

**}**

**});**

print**(**list**);**

**}** **else** **{**

Collections**.**sort**(**list**,** **new** Comparator**<**Stu**>()** **{**

@Override

public int compare**(**Stu o1**,** Stu o2**)** **{**

int dif **=** o2**.**score **-** o1**.**score**;**

**if** **(**dif **==** 0**)** **{**

dif **=** o1**.**num **-** o2**.**num**;**

**}**

**return** dif**;**

**}**

**});**

print**(**list**);**

**}**

**}**

scanner**.**close**();**

**}**

static void print**(**List**<**Stu**>** list**)** **{**

int i**=**0**;**

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

System**.**out**.**println**(**list**.**get**(**i**));**

**}**

**}**

**}**

class Stu **{**

String name**;**

int num**;**

int score**;**

public Stu**(**String na**,** int n**,** int s**)** **{**

name **=** na**;**

num **=** n**;**

score **=** s**;**

**}**

@Override

public String toString**()** **{**

// TODO Auto-generated method stub

**return** name **+** " " **+** score**;**

**}**

**}**

# 矩阵乘法计算量估算

|  |  |
| --- | --- |
| 描述 | 矩阵乘法的运算量与矩阵乘法的顺序强相关。  例如：      A是一个50×10的矩阵，B是10×20的矩阵，C是20×5的矩阵    计算A\*B\*C有两种顺序：（（AB）C）或者（A（BC）），前者需要计算15000次乘法，后者只需要3500次。    编写程序计算不同的计算顺序需要进行的乘法次数 |
| 知识点 | 字符串 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入多行，先输入要计算乘法的矩阵个数n，每个矩阵的行数，列数，总共2n的数，最后输入要计算的法则  3       //矩阵个数n  50 10   //矩阵A的行数50，列数10 10 20   //矩阵B的行数10，列数20 20 5    //矩阵C的行数20，列数5 (A(BC)) //矩阵从A开始命名，A、B、C、D...以此类推，通过括号表示运算顺序 |
| 输出 | 输出需要进行的乘法次数 |
| 样例输入 | 3 50 10 10 20 20 5 (A(BC)) |
| 样例输出 | 3500 |

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

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

int**[][]** matrix **=** **new** int**[**27**][**2**];**

int index **=** 0**,** sum **=** 0**;**

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

matrix**[**i**][**0**]** **=** scanner**.**nextInt**();**

matrix**[**i**][**1**]** **=** scanner**.**nextInt**();**

**}**

String fun **=** scanner**.**next**();**

// print2(matrix);

// System.out.println(fun);

//2个矩阵 m\*n n\*p,其乘法次数为m\*n\*p,相乘后得到的矩阵为m\*p

MyStack stack **=** **new** MyStack**();**

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

char ch **=** fun**.**charAt**(**i**);**

**if** **(**ch **!=** ')'**)** **{**

stack**.**push**(**ch**);**

**}** **else** **{**

//提取（）内的矩阵

ArrayList**<**Character**>** list **=** **new** ArrayList**<>();**

**do** **{**

Character get **=** stack**.**pop**();**

**if** **(**get **==** '('**)** **{**

**break;**

**}** **else** **if** **(**get **!=** '('**)** **{**

list**.**add**(**get**);**

**}**

**}** **while(true);**

//对（）内的矩阵从左到右依次相乘

int id **=** list**.**size**()-**1**;**

char mult1 **=** list**.**get**(**id**--);**//第一个矩阵

char mult2 **=** list**.**get**(**id**--);**//第二个矩阵

sum **+=** matrix**[**mult1**-**'A'**][**0**]** //乘法次数累计

**\***matrix**[**mult1**-**'A'**][**1**]\***matrix**[**mult2**-**'A'**][**1**];**

int row **=** matrix**[**mult1**-**'A'**][**0**];**

int col **=** matrix**[**mult2**-**'A'**][**1**];**

//矩阵个数大于2

**while** **(**id **>=** 0**)** **{**

char mult **=** list**.**get**(**id**--);**

sum **+=** row**\***col**\***matrix**[**mult**-**'A'**][**1**];**

row **=** row**;**

col **=** matrix**[**mult**-**'A'**][**1**];**

**}**

//存入新的矩阵，借用之前矩阵数据的存放位置

matrix**[**mult1**-**'A'**][**0**]** **=** row**;**

matrix**[**mult1**-**'A'**][**1**]** **=** col**;**

stack**.**push**(**mult1**);**

**}**

**}**

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

**}**

scanner**.**close**();**

**}**

static void print2**(**int**[][]** a**)** **{**

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

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

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

**}**

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

**}**

**}**

**}**

class MyStack **{**

char**[]** data **=** **new** char**[**1024**];**

int top **=** **-**1**;**

public void push**(**char ch**)** **{**

top**++;**

data**[**top**]** **=** ch**;**

**}**

public Character pop**()** **{**

Character get**=null;**

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

get **=** data**[**top**];**

top**--;**

**}**

**return** get**;**

**}**

@Override

public String toString**()** **{**

// TODO Auto-generated method stub

**return** String**.**valueOf**(**data**).**substring**(**0**,** top**+**1**);**

**}**

**}**

# 参数解析

|  |  |
| --- | --- |
| 描述 | 在命令行输入如下命令：  xcopy /s c:\ d:\，  各个参数如下：  参数1：命令字xcopy  参数2：字符串/s  参数3：字符串c:\  参数4: 字符串d:\  请编写一个参数解析程序，实现将命令行各个参数解析出来。    解析规则：  1.参数分隔符为空格  2.对于用“”包含起来的参数，如果中间有空格，不能解析为多个参数。比如在命令行输入xcopy /s “C:\program files” “d:\”时，参数仍然是4个，第3个参数应该是字符串C:\program files，而不是C:\program，注意输出参数时，需要将“”去掉，引号不存在嵌套情况。 3.参数不定长  4.输入由用例保证，不会出现不符合要求的输入 |
| 知识点 | 字符串 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入一行字符串，可以有空格 |
| 输出 | 输出参数个数，分解后的参数，每个参数都独占一行 |
| 样例输入 | xcopy /s c:\\ d:\\ |
| 样例输出 | 4 xcopy /s c:\\ d:\\ |

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

int**[]** ms **=** **{**0**,**31**,**28**,**31**,** 30**,**31**,**30**,** 31**,**31**,**30**,** 31**,**30**,**31**};**

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

**while** **(**scanner**.**hasNext**())** **{**

String in **=** scanner**.**nextLine**();**

List**<**String**>** list **=** **new** ArrayList**<>();**

StringBuilder sb **=** **new** StringBuilder**();**

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

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

char ch **=** in**.**charAt**(**i**);**

**if** **(**ch **==** '"'**)** **{**

StringBuilder sb2 **=** **new** StringBuilder**();**

sb2**.**append**(**ch**);**

**do** **{**

i**++;**

ch **=** in**.**charAt**(**i**);**

**if** **(**ch **==** '"'**)** **{**

sb2**.**append**(**ch**);**

i**++;**

**break;**

**}** **else** **{**

sb2**.**append**(**ch**);**

**}**

**}** **while** **(true);**

list**.**add**(**sb2**.**toString**());**

**continue;**

**}**

**if** **(**Character**.**isWhitespace**(**ch**))** **{**

**if** **(**sb**.**length**()** **>** 0**)** **{**

list**.**add**(**sb**.**toString**());**

sb**.**delete**(**0**,** sb**.**length**());**

**}**

**}** **else** **{**

sb**.**append**(**ch**);**

**}**

**if** **(**i **==** in**.**length**()-**1**)** **{**

**if** **(**sb**.**length**()** **>** 0**)** **{**

list**.**add**(**sb**.**toString**());**

sb**.**delete**(**0**,** sb**.**length**());**

**}**

**}**

**}**

System**.**out**.**println**(**list**.**size**());**

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

System**.**out**.**println**(**list**.**get**(**i**));**

**}**

**}**

scanner**.**close**();**

**}**

**}**

# 24点运算

|  |  |
| --- | --- |
| 描述 | 计算24点是一种扑克牌益智游戏，随机抽出4张扑克牌，通过加(+)，减(-)，乘(\*), 除(/)四种运算法则计算  得到整数24，本问题中，扑克牌通过如下字符或者字符串表示，其中，小写joker表示小王，  大写JOKER表示大王：                     3 4 5 6 7 8 9 10 J Q K A 2 joker JOKER  本程序要求实现：输入4张牌，输出一个算式，算式的结果为24点。  详细说明：  1.运算只考虑加减乘除运算，没有阶乘等特殊运算符号，友情提醒，整数除法要当心；  2.牌面2~10对应的权值为2~10, J、Q、K、A权值分别为为11、12、13、1；  3.输入4张牌为字符串形式，以一个空格隔开，首尾无空格；如果输入的4张牌中包含大小王，则输出字符串“ERROR”，  表示无法运算；  4.输出的算式格式为4张牌通过+-\*/四个运算符相连，中间无空格，4张牌出现顺序任意，只要结果正确；  5.输出算式的运算顺序从左至右，不包含括号，如1+2+3\*4的结果为24  6.如果存在多种算式都能计算得出24，只需输出一种即可，如果无法得出24，则输出“NONE”表示无解。 |
| 知识点 | 字符串,循环,函数,指针,枚举,位运算,结构体,联合体,文件操作,递归 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入4张牌为字符串形式，以一个空格隔开，首尾无空格； |
| 输出 | 如果输入的4张牌中包含大小王，则输出字符串“ERROR”，表示无法运算；  输出的算式格式为4张牌通过+-\*/四个运算符相连，中间无空格，4张牌出现顺序任意，  只要结果正确；  输出算式的运算顺序从左至右，不包含括号，如1+2+3\*4的结果为24  如果存在多种算式都能计算得出24，只需输出一种即可，如果无法得出24，  则输出“NONE”表示无解。 |
| 样例输入 | A A A A |
| 样例输出 | NONE |

**import** java**.**util**.**Scanner**;**

public class Main **{**

static int**[]** cards**;**

final static int N **=** 4**;**//N张牌

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

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

**while** **(**scanner**.**hasNext**())** **{**

String in **=** scanner**.**nextLine**();**

int**[]** book **=** **new** int**[**N**];**

cards **=** parseCard**(**in**,** N**);**

**if** **(**cards **==** **null)** **{**

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

**continue;**

**}**

int**[]** arr **=** **new** int**[**N**];**

dfsSequence**(**arr**,** book**,** 0**);**

**if** **(!**findFlag**)** **{**

System**.**out**.**println**(**"NONE"**);**

**}**

**}**

scanner**.**close**();**

**}**

//获取输入牌的值 in:输入纸牌字符串 n张牌

static int**[]** parseCard**(**String in**,** int n**)** **{**

String**[]** cards **=** in**.**split**(**" "**);**

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

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

**if** **(**cards**[**i**].**equals**(**"joker"**)** **||** cards**[**i**].**equals**(**"JOKER"**))** **{**

**return** **null;**

**}**

**if** **(**cards**[**i**].**equals**(**"A"**))** **{**

values**[**i**]** **=** 1**;**

**}** **else** **if** **(**cards**[**i**].**equals**(**"J"**))** **{**

values**[**i**]** **=** 11**;**

**}** **else** **if** **(**cards**[**i**].**equals**(**"Q"**))** **{**

values**[**i**]** **=** 12**;**

**}** **else** **if** **(**cards**[**i**].**equals**(**"K"**))** **{**

values**[**i**]** **=** 13**;**

**}** **else** **{**

values**[**i**]** **=** Integer**.**parseInt**(**cards**[**i**]);**

**}**

**}**

**return** values**;**

**}**

//对N张牌进行排列组合 共有N!种可能

static void dfsSequence**(**int**[]** arr**,** int**[]** book**,** int id**)** **{**

**if** **(**findFlag**)** **return;**

**if** **(**id **==** arr**.**length**)** **{**

// print(arr);

dfsCal**(**arr**,** arr**[**0**],** 1**,** ""**);**

**return;**

**}**

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

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

book**[**i**]** **=** 1**;**

arr**[**id**]** **=** cards**[**i**];**

dfsSequence**(**arr**,** book**,** id**+**1**);**

arr**[**id**]** **=** 0**;**

book**[**i**]** **=** 0**;**

**}**

**}**

**}**

//对N张牌的每一种排列组合进行 + - 乘除运算，每一种组合有4^(N-1)种运算

//最多共有 N! \* 4^(N-1) 种运算方式

//初始值res为数组中的第一个元素，id为1,即下一个运算值的下标

static boolean findFlag **=** **false;**

static void dfsCal**(**int**[]** arr**,** int res**,** int id**,** String symbol**)** **{**

**if** **(**findFlag**)** **return;**

**if** **(**id **==** arr**.**length**)** **{**

**if** **(**res **==** 24**)** **{**

StringBuilder sb **=** **new** StringBuilder**();**

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

sb**.**append**(**arr**[**i**]).**append**(**symbol**.**charAt**(**i**));**

**}**

sb**.**append**(**arr**[**arr**.**length**-**1**]);**

// sb.append(arr[0]).append(symbol.charAt(0)).

// append(arr[1]).append(symbol.charAt(1)).

// append(arr[2]).append(symbol.charAt(2)).

// append(arr[3]);

System**.**out**.**println**(**sb**.**toString**());**

findFlag **=** **true;**

**}**

**return;**

**}**

int resNew**;**

//+

resNew **=** res **+** arr**[**id**];**

dfsCal**(**arr**,** resNew**,** id**+**1**,** symbol**+**"+"**);**

//-

resNew **=** res **-** arr**[**id**];**

dfsCal**(**arr**,** resNew**,** id**+**1**,** symbol**+**"-"**);**

//\*

resNew **=** res **\*** arr**[**id**];**

dfsCal**(**arr**,** resNew**,** id**+**1**,** symbol**+**"\*"**);**

// / 判断是否可整除

**if** **(**res **%** arr**[**id**]** **==** 0**)** **{**

resNew **=** res **/** arr**[**id**];**

dfsCal**(**arr**,** resNew**,** id**+**1**,** symbol**+**"/"**);**

**}**

**}**

static void print**(**int**[]** a**)** **{**

**if** **(**a **==** **null)** **{**

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

**return;**

**}**

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

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

**}**

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

**}**

**}**

# JAVA 题目2-3级

|  |  |
| --- | --- |
| 描述 | 请编写一个函数（允许增加子函数），计算n x m的棋盘格子（n为横向的格子数，m为竖向的格子数）沿着各自边缘线从左上角走到右下角，总共有多少种走法，要求不能走回头路，即：只能往右和往下走，不能往左和往上走。 |
| 知识点 | 字符串,循环,函数,指针,枚举,位运算,结构体,联合体,文件操作,递归 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入两个正整数 |
| 输出 | 返回结果 |
| 样例输入 | 2 2 |
| 样例输出 | 6 |

package com**.**oj**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

static int paths **=** 0**;**

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

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

**while** **(**scanner**.**hasNext**())** **{**

int m **=** scanner**.**nextInt**();**

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

int**[][]** map **=** **new** int**[**m**+**1**][**n**+**1**];**

paths **=** 0**;**

getPaths**(**map**,** 0**,** 0**);**

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

**}**

scanner**.**close**();**

**}**

static void getPaths**(**int**[][]** map**,** int x**,** int y**)** **{**

**if** **(**x **==** map**.**length**-**1 **&&** y **==** map**[**0**].**length **-**1**)** **{**

paths**++;**

// map[x][y] = 1;

// print(map);

// map[x][y] = 0;

// System.out.println();

**return;**

**}**

map**[**x**][**y**]** **=** 1**;**

int tx **=** x **+** 1**;**

int ty **=** y**;**

**if** **(**tx **>=** 0 **&&** tx **<** map**.**length **&&** ty **>=** 0 **&&** ty **<** map**[**0**].**length**)** **{**

getPaths**(**map**,** tx**,** ty**);**

**}**

tx **=** x**;**

ty **=** y **+** 1**;**

**if** **(**tx **>=** 0 **&&** tx **<** map**.**length **&&** ty **>=** 0 **&&** ty **<** map**[**0**].**length**)** **{**

getPaths**(**map**,** tx**,** ty**);**

**}**

map**[**x**][**y**]** **=** 0**;**

**}**

static void print**(**int**[][]** a**)** **{**

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

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

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

**}**

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

**}**

**}**

**}**

另外的解题思路：其实是求C（m+n）n

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

int m **=** scanner**.**nextInt**();**

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

// System.out.println(getPath(m, n));

System**.**out**.**println**(**getPath2**(**m**,** n**));**

**}**

scanner**.**close**();**

**}**

//递归

static int getPath**(**int m**,** int n**)** **{**

**if** **(**m **==** 0 **||** n **==** 0**)** **{**

**return** 1**;**

**}**

**if** **(**m **>** 0 **&&** n **>** 0**)** **{**

**return** getPath**(**m**,** n**-**1**)** **+** getPath**(**m**-**1**,** n**);**

**}**

**return** 0**;**

**}**

//dp

static int getPath2**(**int m**,** int n**)** **{**

int**[][]** dp **=** **new** int**[**m**+**1**][**n**+**1**];**

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

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

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

dp**[**i**][**j**]** **=** 1**;**

**continue;**

**}**

dp**[**i**][**j**]** **=** dp**[**i**-**1**][**j**]** **+** dp**[**i**][**j**-**1**];**

**}**

**}**

**return** dp**[**m**][**n**];**

**}**

**}**

# Redraiment的走法

|  |  |
| --- | --- |
| 描述 | 题目描述     Redraiment是走梅花桩的高手。Redraiment总是起点不限，从前到后，往高的桩子走，但走的步数最多，不知道为什么？你能替Redraiment研究他最多走的步数吗？    样例输入  6  2 5 1 5 4 5    样例输出  3    提示  Example:  6个点的高度各为 2 5 1 5 4 5  如从第1格开始走,最多为3步, 2 4 5  从第2格开始走,最多只有1步,5  而从第3格开始走最多有3步,1 4 5  从第5格开始走最多有2步,4 5  所以这个结果是3。    接口说明  方法原型：      int GetResult(int num, int[] pInput, List  pResult);  输入参数：    int num：整数，表示数组元素的个数（保证有效）。    int[] pInput: 数组，存放输入的数字。  输出参数：    List pResult: 保证传入一个空的List，要求把结果放入第一个位置。 返回值：   正确返回1，错误返回0 |
| 知识点 | 排序 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入多行，先输入数组的个数，再输入相应个数的整数 |
| 输出 | 输出结果 |
| 样例输入 | 6 2 5 1 5 4 5 |
| 样例输出 | 3 |

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

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

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

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

arr**[**i**]** **=** scanner**.**nextInt**();**

**}**

System**.**out**.**println**(**getMaxLen**(**arr**));**

**}**

scanner**.**close**();**

**}**

static int getMaxLen**(**int**[]** arr**)** **{**

int len **=** arr**.**length**;**

int max **=** 0**;**

//dp[i][j]表示以下标i元素为起点

//到j元素之间最多可以有多少个递增元素

//i>j

int**[][]** dp **=** **new** int**[**len**][**len**];**

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

**for** **(**int n**=**m**;** n**<**len**;** n**++)** **{**

//初始化

**if** **(**m **==** n**)** **{**

dp**[**m**][**n**]** **=** 1**;**

**continue;**

**}**

**//对左边元素查找，若小于当前元素且大于起点元素，则判断**

int k **=** m**;**

**while** **(**k **<** n**)** **{**

**if** **(**arr**[**k**]** **<** arr**[**n**]** **&&** arr**[**k**]** **>=** arr**[**m**])** **{**

dp**[**m**][**n**]** **=** max**(**dp**[**m**][**n**],** dp**[**m**][**k**]** **+** 1**);**

**}**

k**++;**

**}**

**if** **(**max **<** dp**[**m**][**n**])** **{**

max **=** dp**[**m**][**n**];**

**}**

**}**

**}**

// print(dp);

**return** max**;**

**}**

private static int max**(**int i**,** int j**)** **{**

// TODO Auto-generated method stub

**return** i **>** j **?** i **:** j**;**

**}**

static void print**(**int**[][]** a**)** **{**

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

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

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

**}**

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

**}**

**}**

**}**

# 字符串合并处理

|  |  |
| --- | --- |
| 描述 | 按照指定规则对输入的字符串进行处理。  详细描述：  将输入的两个字符串合并。  对合并后的字符串进行排序，要求为：下标为奇数的字符和下标为偶数的字符分别从小到大排序。这里的下标意思是字符在字符串中的位置。  对排训后的字符串进行操作，如果字符为‘0’——‘9’或者‘A’——‘F’或者‘a’——‘f’，则对他们所代表的16进制的数进行BIT倒序的操作，并转换为相应的大写字符。如字符为‘4’，为0100b，则翻转后为0010b，也就是2。转换后的字符为‘2’； 如字符为‘7’，为0111b，则翻转后为1110b，也就是e。转换后的字符为大写‘E’。    举例：输入str1为"dec"，str2为"fab"，合并为“decfab”，分别对“dca”和“efb”进行排序，排序后为“abcedf”，转换后为“5D37BF”  接口设计及说明：  /\*  功能:字符串处理  输入:两个字符串,需要异常处理  输出:合并处理后的字符串，具体要求参考文档  返回:无  \*/  void ProcessString(char\* str1,char \*str2,char \* strOutput)  {  } |
| 知识点 | 字符串,排序,位运算 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 输入两个字符串 |
| 输出 | 输出转化后的结果 |
| 样例输入 | dec fab |
| 样例输出 | 5D37BF |

注意题目中没说的，有可能字符会不在0-F范围内比如会出现G,H之类的，这种类型的字符也参与排序，但不参与编码，即在0-F（及0-f）范围之外的字符保留原样

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Arrays**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

String s1 **=** scanner**.**next**();**

String s2 **=** scanner**.**next**();**

// String ss = change(s1+s2);

System**.**out**.**println**(**transfer**(**s1**+**s2**));**

**}**

scanner**.**close**();**

**}**

static String transfer**(**String str**)** **{**

//奇偶排序

char**[]** cs **=** str**.**toCharArray**();**

int len **=** str**.**length**();**

// int k=0;

// while (k <= 1) {//k=0 偶排序， k=1奇排序

// //选择排序

// for (int i=k; i<len-1; i+=2) {

// int m = i;

// for (int j=i+2; j<len; j+=2) {

// if (cs[m] > cs[j]) {

// m = j;

// }

// }

// if (m != i) {

// char t = cs[i];

// cs[i] = cs[m];

// cs[m] = t;

// }

// }

// k++;

// }

int k2 **=** 2**;**

**while** **(**k2 **<=** 3**)** **{**

//插入排序

**for** **(**int i**=**k2**;** i**<**len**;** i**+=**2**)** **{**

int m **=** i**;**

**while(**m**-**2 **>=** 0**)** **{**

**if** **(**cs**[**m**-**2**]** **>** cs**[**m**])** **{**

char t **=** cs**[**m**-**2**];**

cs**[**m**-**2**]** **=** cs**[**m**];**

cs**[**m**]** **=** t**;**

**}** **else** **{**

**break;**

**}**

m**-=**2**;**

**}**

**}**

k2**++;**

**}**

//转换

StringBuilder sb2 **=** **new** StringBuilder**();**

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

sb2**.**append**(**reverse**(**cs**[**i**]));**

**}**

// System.out.println(String.valueOf(cs));

**return** **(**sb2**.**toString**());**

**}**

//奇偶拆分 排序 合并 转换

static String change**(**String s1**)** **{**

//奇偶拆分

char**[]** odd **=** **new** char**[**s1**.**length**()];**

char**[]** even **=** **new** char**[**s1**.**length**()];**

int len **=** s1**.**length**(),** ie**=**0**,**io**=**0**;**

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

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

even**[**ie**++]** **=** s1**.**charAt**(**i**);**

**}** **else** **{**

odd**[**io**++]** **=** s1**.**charAt**(**i**);**

**}**

**}**

// System.out.println(String.valueOf(even));

// System.out.println(String.valueOf(odd));

//排序

Arrays**.**sort**(**even**,** 0**,** ie**);**

Arrays**.**sort**(**odd**,** 0**,** io**);**

StringBuilder sb **=** **new** StringBuilder**();**

//合并

ie**=**0**;**io**=**0**;**

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

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

sb**.**append**(**even**[**ie**++]);**

**}** **else** **{**

sb**.**append**(**odd**[**io**++]);**

**}**

**}**

//转换

StringBuilder sb2 **=** **new** StringBuilder**();**

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

sb2**.**append**(**reverse**(**sb**.**charAt**(**i**)));**

**}**

**return** **(**sb2**.**toString**());**

**}**

static char reverse**(**char ch**)** **{**

char c **=** Character**.**toLowerCase**(**ch**);**

**if** **(!((**c**>=**'0' **&&** c**<=**'9'**)** **||** **(**c **>=** 'a' **&&** c **<=** 'f'**)))** **{**

**return** ch**;**

**}**

int n **=** 0**;**

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

n **=** c **-** '0'**;**

**}**

**if** **(**c **>=** 'a' **&&** c **<=** 'f'**)** **{**

n **=** c **-** 'a' **+** 10**;**

**}**

StringBuilder s1 **=** **new** StringBuilder**(**get4Bin**(**n**));**

String s2 **=** s1**.**reverse**().**toString**();**

int n2 **=** Integer**.**parseInt**(**s2**,** 2**);**

**if** **(**n2 **>=** 0 **&&** n2 **<=** 9**)** **{**

c **=** **(**char**)** **(**n2 **+** '0'**);**

**}**

**if** **(**n2 **>=** 10 **&&** n2 **<=** 15**)** **{**

c **=** **(**char**)** **(**n2 **-** 10 **+** 'a'**);**

**}**

// System.out.println(n2);

**return** Character**.**toUpperCase**(**c**);**

**}**

static String get4Bin**(**int n**)** **{**

String str **=** Integer**.**toBinaryString**(**n**);**

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

**if** **(**str**.**length**()** **<** 4**)** **{**

int i **=** 4 **-** str**.**length**();**

**while** **(**i**--** **>** 0**)** **{**

sb**.**insert**(**0**,** '0'**);**

**}**

**}**

**return** sb**.**toString**();**

**}**

**}**

# 一笔画判断

|  |  |
| --- | --- |
| 描述 | 一笔画游戏是一个数学游戏  即平面上由多条线段构成的一个图形能不能一笔画成，使得在每条线段上都不重复？例如汉字‘日’和‘中’字都可以一笔画的，而‘田’和‘目’则不能。  请编程实现一笔画：  首先输入坐标系上的点数个数，然后输入点的坐标，请判断这张图是否可以一笔画出，并输出画线顺序 （每条线段必须经过一次，且只能经过一次。每个端点可以经过多次。）  **当有多种方式可以完成一笔画时，每一步都必须尽可能先画数值最小的端点。**  如  输入 1,2  1,3 2，3 三个点  http://career-oj.huawei.com/exam/manager/uploadImage/1400832164178.png  画线顺序为：1 2 3 1  如果能画线输出true  其他输出false； |
| 知识点 | 链表,队列,栈,树,图,数组 |
| 运行时间限制 | 10M |
| 内存限制 | 128 |
| 输入 | 1、输入点的个数  2、输入点的坐标 |
| 输出 | 如果可以一笔画这输出 画线顺序 |
| 样例输入 | 3 1 2 1 3 2 3 |
| 样例输出 | true |

只有125分的代码：

package com**.**oj**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Scanner**;**

**import** java**.**util**.**TreeSet**;**

//三角形 3 1 2 1 3 2 3

//日字 7 1 2 1 3 2 4 3 4 3 5 4 6 5 6

//中 9 1 3 2 3 3 4 4 7 7 6 6 5 5 2 6 8 3 6

//目字 10 1 2 1 3 2 4 3 4 3 5 4 6 5 6 5 7 6 8 7 8

public class Main **{**

static int Edges **=** 0**,**Nodes**=**0**;**//总的边数，顶点数

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

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

**while** **(**scanner**.**hasNext**())** **{**

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

Edges **=** n**;**

TreeSet**<**Integer**>** set **=** **new** TreeSet**<>();**

ArrayList**<**Integer**>** list **=** **new** ArrayList**<>();**

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

int a **=** scanner**.**nextInt**();**

int b **=** scanner**.**nextInt**();**

list**.**add**(**a**);**

list**.**add**(**b**);**

set**.**add**(**a**);**

set**.**add**(**b**);**

**}**

Nodes **=** set**.**size**();**

int**[][]** map **=** **new** int**[**Nodes**+**1**][**Nodes**+**1**];**

int**[][]** book **=** **new** int**[**Nodes**+**1**][**Nodes**+**1**];**

**for** **(**int i**=**0**;** i**<**list**.**size**();** i**+=**2**)** **{**

int j **=** i**+**1**;**

map**[**list**.**get**(**i**)][**list**.**get**(**j**)]** **=** 1**;**

map**[**list**.**get**(**j**)][**list**.**get**(**i**)]** **=** 1**;**

**}**

// print(map);

enableFlag **=** **false;**

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

dfs**(**map**,** book**,** i**,** 0**,** ""**);**

**}**

// dfs(map, book, 3, 0);

**if** **(**enableFlag**)** **{**

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

// System.out.println(out);

**}** **else** **{**

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

**}**

**}**

scanner**.**close**();**

**}**

static boolean enableFlag **=** **false;**

static String out**=**""**;**

//node表示走到当前节点编号，bookNodes表示走过了几个顶点

static void dfs**(**int**[][]** map**,** int**[][]** book**,** int node**,** int edges**,** String path**)** **{**

**if** **(**enableFlag**)** **return;**

**if** **(**edges **==** Edges**)** **{**

enableFlag **=** **true;**

// System.out.println(path+node);

out **=** path**+**node**;**

**return;**

**}**

//从顶点node 到顶点 i

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

**if** **(**map**[**node**][**i**]** **==** 1 **&&** book**[**node**][**i**]** **==** 0**)** **{**

book**[**node**][**i**]** **=** 1**;**

book**[**i**][**node**]** **=** 1**;**

dfs**(**map**,** book**,** i**,** edges**+**1**,** path**+**node**+**" "**);**

book**[**node**][**i**]** **=** 0**;**

book**[**i**][**node**]** **=** 0**;**

**}**

**}**

**}**

static void print**(**int**[][]** a**)** **{**

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

**for** **(**int j**=**1**;**j**<**a**[**0**].**length**;** j**++)** **{**

System**.**out**.**printf**(**"%d "**,**a**[**i**][**j**]);**

**}**

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

**}**

**}**

**}**

# 扑克牌大小

|  |  |
| --- | --- |
| 描述 | 扑克牌游戏大家应该都比较熟悉了，一副牌由54张组成，含3~A、2各4张，小王1张，大王1张。牌面从小到大用如下字符和字符串表示（其中，小写joker表示小王，大写JOKER表示大王）： 3 4 5 6 7 8 9 10 J Q K A 2 joker JOKER 输入两手牌，两手牌之间用"-"连接，每手牌的每张牌以空格分隔，"-"两边没有空格，如：4 4 4 4-joker JOKER。 请比较两手牌大小，输出较大的牌，如果不存在比较关系则输出ERROR。 基本规则： （1）输入每手牌可能是个子、对子、顺子（连续5张）、三个、炸弹（四个）和对王中的一种，不存在其他情况，由输入保证两手牌都是合法的，顺子已经从小到大排列； （2）除了炸弹和对王可以和所有牌比较之外，其他类型的牌只能跟相同类型的存在比较关系（如，对子跟对子比较，三个跟三个比较），不考虑拆牌情况（如：将对子拆分成个子）； （3）大小规则跟大家平时了解的常见规则相同，个子、对子、三个比较牌面大小；顺子比较最小牌大小；炸弹大于前面所有的牌，炸弹之间比较牌面大小；对王是最大的牌；  （4）输入的两手牌不会出现相等的情况。 |
| 知识点 | 字符串,循环,链表,队列,栈,查找,搜索,排序,树,图,数组,函数,指针,枚举,位运算,结构体,联合体,文件操作,递归 |
| 运行时间限制 | 0M |
| 内存限制 | 0 |
| 输入 | 输入两手牌，两手牌之间用"-"连接，每手牌的每张牌以空格分隔，"-"两边没有空格，如 4 4 4 4-joker JOKER。 |
| 输出 | 输出两手牌中较大的那手，不含连接符，扑克牌顺序不变，仍以空格隔开；如果不存在比较关系则输出ERROR。 |
| 样例输入 | 4 4 4 4-joker JOKER |
| 样例输出 | joker JOKER |

**import** java**.**awt**.**print**.**Printable**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**HashMap**;**

**import** java**.**util**.**LinkedList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

public class Main **{**

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

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

**while** **(**scanner**.**hasNext**())** **{**

boolean bigLarger **=** **false;**//牌多的一组大的标志位

String**[]** cards **=** scanner**.**nextLine**().**split**(**"-"**);**

String**[]** t1 **=** cards**[**0**].**split**(**" "**);**

String**[]** t2 **=** cards**[**1**].**split**(**" "**);**

String**[]** big **=** t1**.**length //牌多的一组

**>** t2**.**length **?** t1 **:** t2**;**

String**[]** small **=** t1**.**length //牌少的一组

**<=** t2**.**length **?** t1 **:** t2**;**

LinkedList**<**String**>** list **=** **(**LinkedList**<**String**>)** init**();**

//牌数相同 1-1 2-2 3-3 4-4 5-5

**if** **(**big**.**length **==** small**.**length**)** **{**

int a **=** list**.**indexOf**(**big**[**0**]);**

int b **=** list**.**indexOf**(**small**[**0**]);**

**if** **(**a **>** b**)** **{**

bigLarger **=** **true;**

**}** **else** **{**

bigLarger **=** **false;**

**}**

**}** **else** **if** **(**big**.**length **==** 2 //双王

**&&** big**[**0**].**toLowerCase**().**equals**(**"joker"**))** **{**

bigLarger **=** **true;**

**}** **else** **if** **(**small**.**length **==** 2 //双王

**&&** small**[**0**].**toLowerCase**().**equals**(**"joker"**))** **{**

bigLarger **=** **false;**

**}** **else** **if** **(**big**.**length **==** 4**)** **{**//炸弹4-1 4-2 4-3

bigLarger **=** **true;**

**}** **else** **if** **(**small**.**length **==** 4**)** **{**//炸弹 4-5

bigLarger **=** **false;**

**}** **else** **{**//无效组合

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

**continue;**

**}**

**if** **(**bigLarger**)** **{**

print**(**big**);**

**}** **else** **{**

print**(**small**);**

**}**

**}**

scanner**.**close**();**

**}**

static List**<**String**>** init**()** **{**

List**<**String**>** list **=** **new** LinkedList**<>();**

list**.**add**(**"3"**);**

list**.**add**(**"4"**);**

list**.**add**(**"5"**);**

list**.**add**(**"6"**);**

list**.**add**(**"7"**);**

list**.**add**(**"8"**);**

list**.**add**(**"9"**);**

list**.**add**(**"10"**);**

list**.**add**(**"J"**);**

list**.**add**(**"Q"**);**

list**.**add**(**"K"**);**

list**.**add**(**"A"**);**

list**.**add**(**"2"**);**

list**.**add**(**"joker"**);**

list**.**add**(**"JOKER"**);**

**return** list**;**

**}**

static void print**(**String**[]** str**)** **{**

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

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

**}**

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

**}**

**}**

# 火车进站

|  |  |
| --- | --- |
| 描述 | 给定一个正整数N代表火车数量，0<N<10，接下来输入火车入站的序列，一共N辆火车，每辆火车以数字1-9编号。要求以字典序排序输出火车出站的序列号。 |
| 知识点 | 栈 |
| 运行时间限制 | 0M |
| 内存限制 | 0 |
| 输入 | 有多组测试用例，每一组第一行输入一个正整数N（0<N<10），第二行包括N个正整数，范围为1到9。 |
| 输出 | 输出以字典序排序的火车出站序列号，每个编号以空格隔开，每个输出序列换行，具体见sample。 |
| 样例输入 | 3 1 2 3 |
| 样例输出 | 1 2 3 1 3 2 2 1 3 2 3 1 3 2 1 |

解答：

　　其实核心就是一个栈，对于第K个数，在第K个数进栈之前，前面的 K-1 个数要么全部出去了，要么都在栈里面，要么部分在栈里面部分出去了。那么可以假想，在第K个数入栈之前，依次从栈里面出去 0个、1个、2个……栈.size()个，然后把第K个入栈，再对于 K+1个同样实施这样的方法——这就是个递归了。

　　出去了的保存在一个队列里面，没出站的保存在栈里面，最后一辆车处理完了递归结束并输出。

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Iterator**;**

**import** java**.**util**.**LinkedList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

**import** java**.**util**.**Stack**;**

**//in:3 1 2 3**

//out:1 2 3 1 3 2 2 1 3 2 3 1 3 2 1

public class Main **{**

static LinkedList**<**List**<**Integer**>>** lists**;**

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

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

// while (scanner.hasNext()) {

Stack**<**Integer**>** stack **=** **new** Stack**<>();**

lists **=** **new** LinkedList**<>();**

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

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

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

trains**[**i**]** **=** scanner**.**nextInt**();**

**}**

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

next**(**stack**,** list**,** trains**,** 0**);**

Iterator**<**List**<**Integer**>>** it **=** lists**.**iterator**();**

**while** **(**it**.**hasNext**())** **{**

list **=** it**.**next**();**

**for** **(**int i**=**0**;** i**<**list**.**size**()-**1**;** i**++)** **{**

System**.**out**.**print**(**list**.**get**(**i**)+**" "**);**

**}**

System**.**out**.**println**(**list**.**get**(**list**.**size**()-**1**));**

**}**

// }

scanner**.**close**();**

**}**

static void next**(**Stack**<**Integer**>** stack**,** List**<**Integer**>** list**,** int**[]** trains**,** int id**)** **{**

**if** **(**id **==** trains**.**length**)** **{**

**while** **(!**stack**.**isEmpty**())** **{**

list**.**add**(**stack**.**pop**());**

**}**

lists**.**add**(**0**,** list**);**

**return;**

**}**

int size **=** stack**.**size**();**

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

Stack**<**Integer**>** tStack **=** **new** Stack**<>();**

tStack**.**addAll**(**stack**);**

List**<**Integer**>** tList **=** **new** ArrayList**<>(**list**);**

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

tList**.**add**(**tStack**.**pop**());**

**}**

tStack**.**push**(**trains**[**id**]);**

next**(**tStack**,** tList**,** trains**,** id**+**1**);**

**}**

**}**

static void print**(**int**[]** str**)** **{**

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

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

**}**

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

**}**

**}**

**static** **void** dfs(**int**[] source, **int** sid,Stack<Integer> stack, **int**[] out, **int** oid) {

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

**while** (!stack.isEmpty()) {

out[oid++] = stack.pop();

}

System.***out***.println(Arrays.*toString*(out));

**return**;

}

**int** size = stack.size();

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

//出栈i个

Stack<Integer> stack2 = *copyStack*(stack);

**int** id = oid;//oid的值需要保存

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

out[id++] = stack2.pop();

}

stack2.push(source[sid]);

*dfs*(source, sid+1, stack2, out, id);

}

}

**static** Stack<Integer> copyStack(Stack<Integer> cc) {

Stack<Integer> res = **new** Stack<>();

Iterator<Integer> it = cc.iterator();

**while** (it.hasNext()) {

res.push(it.next());

}

**return** res;

}

# 数据分类处理

|  |  |
| --- | --- |
| 描述 | 信息社会，有海量的数据需要分析处理，比如公安局分析身份证号码、QQ用户、手机号码、银行帐号等信息及活动记录。  采集输入大数据和分类规则，通过大数据分类处理程序，将大数据分类输出。 |
| 知识点 | 排序 |
| 运行时间限制 | 0M |
| 内存限制 | 0 |
| 输入 | ﻿一组输入整数序列I和一组规则整数序列R，I和R序列的第一个整数为序列的个数（个数不包含第一个整数）；整数范围为0~0xFFFFFFFF，序列个数不限 |
| 输出 | ﻿从R依次中取出R<i>，对I进行处理，找到满足条件的I<j>：  I<j>整数对应的数字需要连续包含R<i>对应的数字。比如R<i>为23，I<j>为231，那么I<j>包含了R<i>，条件满足 。  按R<i>从小到大的顺序:  (1)先输出R<i>；  (2)再输出满足条件的I<j>的个数；  (3)然后输出满足条件的I<j>在I序列中的位置索引(从0开始)；  (4)最后再输出I<j>。  附加条件：  (1)R<i>需要从小到大排序。相同的R<i>只需要输出索引小的以及满足条件的I<j>，索引大的需要过滤掉  (2)如果没有满足条件的I<j>，对应的R<i>不用输出  (3)最后需要在输出序列的第一个整数位置记录后续整数序列的个数(不包含“个数”本身)    序列I：15,123,456,786,453,46,7,5,3,665,453456,745,456,786,453,123（第一个15表明后续有15个整数）  序列R：5,6,3,6,3,0（第一个5表明后续有5个整数）  输出：*30, 3,6,0,123*,3,453,7,3,9,453456,13,453,14,123,6,7,1,456,2,786,4,46,8,665,9,453456,11,456,12,786  说明：  *30*----后续有30个整数  *3*----从小到大排序，第一个R<i>为0，但没有满足条件的I<j>，不输出0，而下一个R<i>是3  *6---* *存在6个包含3的I<j>*  *0---* *123所在的原序号为0*  *123---* *123包含3，满足条件* |
| 样例输入 | 15 123 456 786 453 46 7 5 3 665 453456 745 456 786 453 123 5 6 3 6 3 0 |
| 样例输出 | 30 3 6 0 123 3 453 7 3 9 453456 13 453 14 123 6 7 1 456 2 786 4 46 8 665 9 453456 11 456 12 786 |

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**HashMap**;**

**import** java**.**util**.**HashSet**;**

**import** java**.**util**.**Iterator**;**

**import** java**.**util**.**LinkedList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Map**;**

**import** java**.**util**.**Scanner**;**

**import** java**.**util**.**Set**;**

**import** java**.**util**.**Stack**;**

**import** java**.**util**.**TreeMap**;**

**import** java**.**util**.**TreeSet**;**

//1 2 3 1 3 2 2 1 3 2 3 1 3 2 1

public class Main **{**

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

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

int m **=** scanner**.**nextInt**();**

String**[]** all **=** **new** String**[**m**];**

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

all**[**i**]** **=** scanner**.**next**();**

**}**

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

Set**<**Integer**>** set **=** **new** TreeSet**<>();**

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

set**.**add**(**scanner**.**nextInt**());**

**}**

Iterator**<**Integer**>** it **=** set**.**iterator**();**

ArrayList**<**Ruler**>** list **=** **new** ArrayList**<>();**

int size **=** 0**;**

**while** **(**it**.**hasNext**())** **{**

String find **=** it**.**next**()+**""**;**

Ruler ruler **=** **new** Ruler**(**find**);**

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

**if** **(**isContain**(**all**[**i**],** find**))** **{**

ruler**.**add**(new** Record**(**i**,** all**[**i**]));**

**}**

**}**

**if** **(**ruler**.**getSize**()** **>** 0**)** **{**

size **+=** ruler**.**getSize**();**

list**.**add**(**ruler**);**

**}**

**}**

**if** **(**size **==** 0**)**

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

**else**

System**.**out**.**print**(**size**+**" "**);**

**for** **(**int i**=**0**;** i**<**list**.**size**()-**1**;** i**++)** **{**

System**.**out**.**print**(**list**.**get**(**i**)+**" "**);**

**}**

System**.**out**.**println**(**list**.**get**(**list**.**size**()-**1**));**

scanner**.**close**();**

**}**

static boolean isContain**(**String str**,** String find**)** **{**

int len **=** find**.**length**();**

**for** **(**int i**=**0**;** i**<=**str**.**length**()-**len**;** i**++)** **{**

**if** **(**str**.**substring**(**i**,** i**+**len**).**equals**(**find**))** **{**

**return** **true;**

**}**

**}**

**return** **false;**

**}**

**}**

class Ruler **{**

String ruler**;**

List**<**Record**>** list**;**

public Ruler**(**String r**)** **{**

ruler **=** r**;**

list **=** **new** ArrayList**<>();**

**}**

public void add**(**Record record**)** **{**

list**.**add**(**record**);**

**}**

public int getSize**()** **{**

**if** **(**list**.**size**()** **>** 0**)**

**return** list**.**size**()\***2 **+** 2**;**

**else**

**return** 0**;**

**}**

@Override

public String toString**()** **{**

StringBuilder sb **=** **new** StringBuilder**();**

sb**.**append**(**ruler**);**

sb**.**append**(**" " **+** list**.**size**());**

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

sb**.**append**(**" "**+**list**.**get**(**i**).**id**+**" "**+**list**.**get**(**i**).**name**);**

**}**

**return** sb**.**toString**();**

**}**

**}**

class Record **{**

int id**;**

String name**;**

public Record**(**int i**,** String n**)** **{**

id **=** i**;**

name **=** n**;**

**}**

**}**

# 配置文件恢复

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 描述 | 有6条配置命令，它们执行的结果分别是：   |  |  | | --- | --- | | **命   令** | **执   行** | | **reset** | reset what | | **reset board** | board fault | | **board add** | where to add | | **board delet** | no board at all | | **reboot backplane** | impossible | | **backplane abort** | install first | | ***he he*** | unkown command |   **注意：he he不是命令。**  为了简化输入，方便用户，以“最短唯一匹配原则”匹配： 1、若只输入一字串，则只匹配一个关键字的命令行。例如输入：r，根据该规则，匹配命令reset，执行结果为：reset what；输入：res，根据该规则，匹配命令reset，执行结果为：reset what； 2、若只输入一字串，但本条命令有两个关键字，则匹配失败。例如输入：reb，可以找到命令reboot backpalne，但是该命令有两个关键词，所有匹配失败，执行结果为：unkown command 3、若输入两字串，则先匹配第一关键字，如果有匹配但不唯一，继续匹配第二关键字，如果仍不唯一，匹配失败。例如输入：r b，找到匹配命令reset board，执行结果为：board fault。  4、若输入两字串，则先匹配第一关键字，如果有匹配但不唯一，继续匹配第二关键字，如果唯一，匹配成功。例如输入：b a，无法确定是命令**board add**还是**backplane abort**，匹配失败。 5、若输入两字串，第一关键字匹配成功，则匹配第二关键字，若无匹配，失败。例如输入：bo a，确定是命令**board add**，匹配成功。 6、若匹配失败，打印“unkonw command” |
| 知识点 | 字符串 |
| 运行时间限制 | 0M |
| 内存限制 | 0 |
| 输入 | 多行字符串，每行字符串一条命令 |
| 输出 | 执行结果，每条命令输出一行 |
| 样例输入 | reset reset board board add board delet reboot backplane backplane abort |
| 样例输出 | reset what board fault where to add no board at all impossible install first |

**import** java**.**util**.**Scanner**;**

//1 2 3 1 3 2 2 1 3 2 3 1 3 2 1

public class Main **{**

static String**[][]** commands **=** **{{**"reset"**,**""**},{**"reset"**,** "board"**}**

**,{**"board"**,** "add"**},{**"board"**,** "delet"**}**

**,{**"reboot"**,** "backplane"**}**

**,{**"backplane"**,** "abort"**}};**

static String**[]** exe **=** **{**"reset what"**,**"board fault"**,** "where to add"

**,**"no board at all"**,** "impossible"**,** "install first"

**,** "unkown command"**};**

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

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

**while** **(**scanner**.**hasNextLine**())** **{**

String in **=** scanner**.**nextLine**();**

System**.**out**.**println**(**getExe**(**in**));**

**}**

scanner**.**close**();**

**}**

static String getExe**(**String reg**)** **{**

int c **=** getMatch**(**reg**);**

**if** **(**c **==** **-**1**)**

**return** exe**[**exe**.**length**-**1**];**

**else**

**return** exe**[**c**];**

**}**

static int getMatch**(**String regexs**)** **{**

int res **=** **-**1**;**

String**[]** regs **=** regexs**.**split**(**" "**);**

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

**if** **(**isMatch**(**commands**[**0**][**0**],** regs**[**0**]))** **{**

res **=** 0**;**

**}**

**}** **else** **if** **(**regs**.**length **==** 2**){**

int i**=**1**,** ts**=**0**;**

**for** **(** **;** i**<**commands**.**length**;** i**++)** **{**

**if** **(**isMatch**(**commands**[**i**][**0**],** regs**[**0**])**

**&&** isMatch**(**commands**[**i**][**1**],** regs**[**1**]))** **{**

ts**++;**

res **=** i**;**

**}**

**}**

**if** **(**ts **>** 1**)** **{**//匹配命令大于1个

res **=** **-**1**;**

**}**

**}**

**return** res**;**

**}**

static boolean isMatch**(**String str**,** String regex**)** **{**

**return** str**.**matches**(**"^"**+**regex**+**".\*"**);**

**}**

**}**

# 二分查找

**static** **boolean** bin\_search(**int**[] arr, **int** value) {

**int** l = 0, r = arr.length;

**int** t = 0;

**while** (r - l >= 1) {

t++;

**int** i = (l + r)/2;

**if** (arr[i] == value) {

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

**return** **true**;

}

**else** **if** (arr[i] < value) l = i + 1;

**else** r = i;

}

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

**return** **false**;

}

# 排序

**package** com.oj;

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

**public** **class** BubbleSort {

//冒泡排序

**public** **static** **int**[] bubbleSort(**int**[] ns, **int** n) {

//better

**for** (**int** j=ns.length-1; j>=1; j--) {

**for** (**int** i=0; i<=j-1; i++) {

**if** (ns[i] > ns[i+1]) {

*swap*(ns, i, i+1);

}

}

}

**return** ns;

}

//选择排序

**public** **static** **int**[] selectionSort(**int**[] arr, **int** n) {

//先找最大

/\*

for (int i=arr.length-1; i>=1; i--) {

int max = i;

for (int j=0; j<=i-1; j++) {

if (arr[max] < arr[j]) {

max = j;

}

}

if (max != i) {

swap(arr, i, max);

}

print(arr);

}

\*/

//先找最小

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

**int** min = i;

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

**if** (arr[min] > arr[j]) {

min = j;

}

}

**if** (min != i) {

*swap*(arr, min, i);

}

// System.out.print(i+1+":");

// print(arr);

}

**return** arr;

}

//插入排序

**public** **static** **int**[] insertionSort(**int**[] arr, **int** n) {

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

**int** insert = i;

**while** (insert > 0) {

**if** (arr[insert] < arr[insert-1]) {

*swap*(arr, insert, insert-1);

insert--;

} **else** {

**break**;

}

}

// print(arr);

}

**return** arr;

}

//归并排序

**public** **static** **int**[] mergeSort(**int**[] arr, **int** n) {

*mergeProcess*(arr, 0, arr.length-1);

**return** arr;

}

**static** **void** mergeProcess(**int**[] arr, **int** left, **int** right) {

**if** (left == right) **return**;

**int** mid = (right + left)/2;

*mergeProcess*(arr, left, mid);

*mergeProcess*(arr, mid+1, right);

*merge*(arr, left, mid, right);

}

//归并排序， mid一般是左边区间的最后一个元素下标

**static** **void** merge(**int**[] arr,**int** left,**int** mid,**int** right) {

**int**[] temp = **new** **int**[right-left+1];

**int** id = 0;

**int** l = left;//左边区间的起始下标

**int** r = mid+1;//右边区间的起始下标

//对2个区间按照大小放入新数组，直到其中一个区间的数据全部放入

**while** (l <= mid && r <= right) {

**if** (arr[l] < arr[r]) {

temp[id++] = arr[l++];

} **else** {

temp[id++] = arr[r++];

}

}

//此时还有一个区间的元素没有被放入组合的数组

//如果是左边区间

**while** (l <= mid) {

temp[id++] = arr[l++];

}

//如果是右边区间

**while** (r <= right) {

temp[id++] = arr[r++];

}

//将原数组待合并的2个区间用新合并的数组覆盖

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

arr[left+i] = temp[i];

}

}

//快速排序

**public** **static** **int**[] quickSort(**int**[] arr, **int** n) {

// qSort(arr, 0, arr.length-1);

*partition*(arr, 0, arr.length-1);

**return** arr;

}

**static** **void** qSort(**int**[] arr, **int** left, **int** right) {

**if** (left < right) {

**int** l = left, r = right;

**int** random = (**int**) (Math.*random*()\*(right-left+1)+left);

*swap*(arr, left, random);

**int** base = arr[left];

**while** (l < r) {

**while** (l < r && arr[r] >= base) {

r--;

}

**while** (l < r && arr[l] <= base) {

l++;

}

**if** (l < r) {

*swap*(arr, r, l);

}

}

arr[left] = arr[l];

arr[l] = base;

*qSort*(arr, left, l-1);

*qSort*(arr, l+1, right);

}

}

**static** **void** partition(**int**[] arr, **int** left,**int** right) {

**if** (left >= right) **return**;

**int** random = (**int**) (Math.*random*()\*(right-left+1) + left);

*swap*(arr, right, random);

**int** mid = left;

**int** index = left;

**while** (index < right) {

**if** (arr[index] < arr[right]) {

*swap*(arr, mid++, index);

}

index++;

}

*swap*(arr, mid, right);

*partition*(arr, left, mid-1);

*partition*(arr, mid+1, right);

}

**static** **void** swap(**int**[] arr, **int** id1,**int** id2) {

**int** temp = arr[id1];

arr[id1] = arr[id2];

arr[id2] = temp;

}

**static** **int**[] generatArray(**int** len, **int** range) {

**int**[] arr = **new** **int**[len];

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

arr[i] = (**int**)(Math.*random*()\*range)+1;

}

**return** arr;

}

**static** **boolean** isSort(**int**[] arr) {

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

**if** (arr[i-1] > arr[i])

**return** **false**;

}

**return** **true**;

}

**private** **static** **void** lookSort(**int** len, **int** range, **int** times) {

**int**[] ns = {10,9,8,7,6,5,4,3,2,1};

System.***out***.print("unsort:");

*print*(ns);

//sort method

// insertionSort(ns, ns.length);

// mergeSort(ns, ns.length);

*quickSort*(ns, ns.length);

System.***out***.print("sorted:");

*print*(ns);

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

}

**static** **boolean** *Debug* = **false**;

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

**int** len = 20,range = 100, times = 50000;

**if** (*Debug*) {

*lookSort*(10, range, 3);

} **else** {

*testSort*(len, range, times);

**int** t = 10;

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

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

**long** start = System.*currentTimeMillis*();

*runTimeOfSort*(1\_000\_000);

**long** end = System.*currentTimeMillis*();

**long** runTime = end - start;

runTimes[i] = (**int**) runTime;

}

**int** sum = 0;

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

sum += runTimes[i];

}

System.***out***.println("run:"+sum/10+"ms");

}

}

**private** **static** **void** testSort(**int** len, **int** range, **int** times) {

**int** i;

**for** (i=0; i<times; i++) {

**int**[] ns = *generatArray*(len, range);

//sort method

// bubbleSort(ns, ns.length);

// selectionSort(ns, ns.length);

// insertionSort(ns, ns.length);

// mergeSort(ns, ns.length);

*quickSort*(ns, ns.length);

**if** (!*isSort*(ns)) {

System.***out***.println("Wrong case:");

*print*(ns);

**break**;

}

}

**if** (i == times)

System.***out***.println("sort is ok");

}

**private** **static** **void** runTimeOfSort(**int** len) {

**int**[] ns = *generatArray*(len, Integer.***MAX\_VALUE***);

//sort method

// bubbleSort(ns, ns.length);

// selectionSort(ns, ns.length);

// insertionSort(ns, ns.length);

// mergeSort(ns, ns.length);

*quickSort*(ns, ns.length);

// Arrays.sort(ns);

}

**static** **void** print(**int**[] n) {

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

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

}

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

}

}

# 递归生成格雷码

**import** java**.**util**.**Scanner**;**

public class Ma **{**

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

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

**while** **(**sc**.**hasNext**())** **{**

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

String**[]** ss **=** getGL**(**n**);**

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

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

**}**

**}**

**}**

static String**[]** getGL**(**int n**)** **{**

String**[]** res **=** **new** String**[(**int**)**Math**.**pow**((**int**)**2**,** **(**int**)**n**)];**

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

res**[**0**]** **=** "0"**;**

res**[**1**]** **=** "1"**;**

**return** res**;**

**}** **else** **{**

String**[]** preStr **=** getGL**(**n**-**1**);**

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

res**[**i**]** **=** "0"**+**preStr**[**i**];**

res**[**i**+**preStr**.**length**]** **=** "1"**+**preStr**[**i**];**

**}**

**return** res**;**

**}**

**}**

**}**

# 主元素问题

**import** java**.**util**.**Scanner**;**

public class Ma **{**

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

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

**while** **(**sc**.**hasNext**())** **{**

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

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

int value **=** 0**,** times **=** 0**;**

//使用一个变量value 和该变量的出现次数times

//当一个值重复出现时，times++，如果是不同值则times--

//如果times减到了0，则用当前值替换value

//这样最终value即为所求出现次数大于一半的值。

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

arr**[**i**]** **=** sc**.**nextInt**();**

**if** **(**times **==** 0**)** **{**

value **=** arr**[**i**];**

times **=** 1**;**

**}** **else** **if** **(**arr**[**i**]** **==** value**)** **{**

times**++;**

**}** **else** **{**

times**--;**

**if** **(**times **==** 0**)** **{**

value **=** arr**[**i**];**

times **=** 1**;**

**}**

**}**

**}**

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

**}**

**}**

**}**

# 二叉树复原

public TreeNode reConstructBinaryTree**(**int **[]** pre**,**int **[]** in**)** **{**

**if** **(**pre **==** **null** **||** in **==** **null)**

**return** **null;**

**return** getRoot**(**pre**,** 0**,** pre**.**length**-**1**,** in**,** 0**,** in**.**length**-**1**);**

**}**

public TreeNode getRoot**(**int**[]** pre**,** int preLeft**,** int preRight**,**

int**[]** in**,** int inLeft**,** int inRight**)** **{**

int rootValue **=** pre**[**preLeft**];**

TreeNode rootNode **=** **new** TreeNode**(**rootValue**);**

**if** **(**preLeft **==** preRight **&&** inLeft **==** inRight **&&** pre**[**preLeft**]** **==** in**[**inLeft**])** **{**

**return** rootNode**;**

**}**

int inRootId **=** 0**;**

**while** **(**inRootId**<=**inRight **&&** in**[**inRootId**]** **!=** rootValue**)** **{**

inRootId**++;**

**}**

int leftLen **=** inRootId **-** inLeft**;**

int preLeftEnd **=** preLeft **+** leftLen**;**

**if** **(**leftLen **>** 0**)** **{**

rootNode**.**left **=** getRoot**(**pre**,** preLeft**+**1**,** preLeftEnd**,** in**,** inLeft**,** inRootId**-**1**);**

**}**

**if** **(**leftLen **<** preRight **-** preLeft**)** **{**

rootNode**.**right **=** getRoot**(**pre**,** preLeftEnd**+**1**,** preRight**,** in**,** inRootId**+**1**,** inRight**);**

**}**

**return** rootNode**;**

**}**

# 小范围排序

已知一个几乎有序的数组，几乎有序是指，如果把数组排好顺序的话，每个元素移动的距离可以不超过k，并且k相对于数组来说比较小。请选择一个合适的排序算法针对这个数据进行排序。

给定一个int数组**A**，同时给定A的大小**n**和题意中的**k**，请返回排序后的数组。

测试样例：

[2,1,4,3,6,5,8,7,10,9],10,2

返回：[1,2,3,4,5,6,7,8,9,10]

public class ScaleSort **{**

//最小的数在0~k-1区域，对此区域生成最小堆，堆顶即为当前最小值，

//将最小值放到数组A[0]中

//然后依次将第k,k+1……等依次放入堆顶，调整堆，并将每次的最小值依次放入数组A[id++]

//最后将堆的数据依次调整放入A中，类似于堆排序的方式。

public int**[]** sortElement**(**int**[]** A**,** int n**,** int k**)** **{**

// write code here

int**[]** heap **=** **new** int**[**k**];**

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

heap**[**i**]** **=** A**[**i**];**

**}**

**for** **(**int i**=**k**/**2**;** i**>=**0**;** i**--)** **{**

heapAdjust**(**heap**,** i**,** k**);**

**}**

int id**=**0**;**

**for** **(**int i**=**k**;** i**<**A**.**length**;** i**++)** **{**

A**[**id**++]** **=** heap**[**0**];**//单纯的覆盖堆顶

heap**[**0**]** **=** A**[**i**];**

heapAdjust**(**heap**,**0**,**k**);**

**}**

**for** **(**int i**=**k**-**1**;** i**>=**0**;** i**--)** **{**

A**[**id**++]** **=** heap**[**0**];**

swap**(**heap**,**0**,**i**);**//和堆的最后位置交换交换

heapAdjust**(**heap**,** 0**,** i**);**

**}**

**return** A**;**

**}**

void heapAdjust**(**int**[]** arr**,** int id**,** int len**)** **{**

**while** **(**id**\***2**+**1**<**len**)** **{**

int child **=** id**\***2**+**1**;**

**if** **(**child**+**1**<**len **&&** arr**[**child**+**1**]** **<** arr**[**child**])**

child**++;**

**if** **(**arr**[**child**]** **<** arr**[**id**])** **{**

swap**(**arr**,** child**,** id**);**

id **=** child**;**

**}** **else** **{**

**break;**

**}**

**}**

**}**

void swap**(**int**[]** arr**,** int i**,** int j**)** **{**

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

arr**[**i**]** **=** arr**[**j**];**

arr**[**j**]** **=** t**;**

**}**

**}**

# 相邻两数最大差值练习题

有一个整形数组A，请设计一个复杂度为O(n)的算法，算出排序后相邻两数的最大差值。

给定一个int数组**A**和**A**的大小**n**，请返回最大的差值。保证数组元素多于1个。

测试样例：

[1,2,5,4,6],5

返回：2

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

public class Gap **{**

public int maxGap**(**int**[]** a**,** int n**)** **{**

// write code here

int len **=** a**.**length**;**

int min **=** a**[**0**],**max**=**a**[**0**];**

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

max **=** Math**.**max**(**max**,** a**[**i**]);**

min **=** Math**.**min**(**min**,** a**[**i**]);**

**}**

int**[]** bucketMax **=** **new** int**[**len**+**1**];**//需要分别存放一个桶中的最大值和最小值

int**[]** bucketMin **=** **new** int**[**len**+**1**];**

boolean**[]** hasNum **=** **new** boolean**[**len**+**1**];**

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

int id **=** getBucketIndex**(**len**,** a**[**i**],** max**,** min**);**

**if** **(**hasNum**[**id**]** **==** **false)** **{**

bucketMax**[**id**]** **=** a**[**i**];**

bucketMin**[**id**]** **=** a**[**i**];**

hasNum**[**id**]** **=** **true;**

**}** **else** **{**

bucketMax**[**id**]** **=** Math**.**max**(**a**[**i**],** bucketMax**[**id**]);**

bucketMin**[**id**]** **=** Math**.**min**(**a**[**i**],** bucketMin**[**id**]);**

**}**

**}**

int maxDif**=**0**;**

int id **=** 0**;**

//当前桶的最小值减去上一个桶的最大值 即为最大差值

**while** **(**id **<** len**+**1**)** **{**

**while(**id **<** len**+**1 **&&** hasNum**[**id**]** **==** **false)** id**++;**

**if** **(**id **==** len**+**1**)break;**

int first **=** bucketMax**[**id**];**

id**++;**//这里递进

**while(**id **<** len**+**1 **&&** hasNum**[**id**]** **==** **false)** id**++;**

**if** **(**id **==** len**+**1**)break;**

int second **=** bucketMin**[**id**];**//这里不需要递进，因为下一次的first是本次的second

maxDif **=** Math**.**max**(**maxDif**,** **(**second**-**first**));**

**}**

**return** maxDif**;**

**}**

int getBucketIndex**(**long len**,**int v**,** int max**,** int min**)** **{**

int id **=(**int**)** **((**v**-**min**)\***len**/(**max**-**min**));**

**return** id**;**

**}**

**}**