**刷题**

# 1 逻辑题

## 1 余数和商

打印N/D的值，若结果是无限循环小数，则小数点后面表示成循环节。eg 1/7=0.（142857）

**知识点**：

* 有理数：一个整数和一个整数的比，有理数的小数部分是有限或为无限循

小数。

* 无理数：小数部分为无限不循环小数。
* 余数相同才才是循环节的终点，商不是。

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

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

System.***out***.println("整数范围均为：1~1000(格式: 整数,整数 )");

String s = scan.nextLine();

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

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

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

System.***out***.println( *f*(m,n));

}

**public** **static** String f(**int** m,**int** n){

StringBuffer sb = **new** StringBuffer(); // 保存结果

List<Integer> lis = **new** ArrayList<Integer>(); // 记录所有余数

String s = m/n+"."; // 保存整数部分

m = m%n; // 得到余数

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

**if**(lis.contains(m)){

**int** i=0; // 得到循环节开始的位置 i

**for**(;i<sb.length();i++){

**if**(sb.charAt(i)-'0'==m\*10/n){ //char类型数据相加减自动转换为int类型。

**break**;

}

}

sb.insert(i,"["); // 为循环节添加"[ ]"

sb.insert(sb.length(),"]");

**break**;

}**else**{

lis.add(m); // 添加余数 1，3，

sb.append(m\*10/n); // 添加商 1，4

}

m = m\*10%n; // 得到下一个余数 3，2

}

**return** sb.insert(0, s).toString();

}

**总结**：

* 两个整数相除：m/n

余数为：r1=m%n 商为：s1=r1\*10/n 下一个余数为：r2=r\*10%n 下一个商为：s2=r2\*10/n

* char类型的相加减会自动转换为int类型。
* 基本类型比较相等时，==判断的是值。

# 2 回溯题

**回溯法**：加了条件的dfs。比如四色问题中限制给a节点必须上b颜色。

## 1地图四色问题

**注意**：

* 只要某一区域的颜色发生改变，那么与它前边所有的区域都要进行比较

方法一

//0表示未着色，1，2，3，4分别表示4种颜色，区域用节点表示，地图用邻接矩阵表示，相邻用1表示，不相邻用0表示。

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

//测试数据

**int** [][]adj=**new** **int** [][]{

{0,1,0,0,0,0,1},

{1,0,1,1,1,1,1},

{0,1,0,1,0,0,0},

{0,1,1,0,1,0,0},

{0,1,0,1,0,1,1},

{0,1,0,0,1,0,1},

{1,1,0,0,1,1,0}

};

**int** []color=**new** **int**[adj.length];

//初始化color数组，0表示未着色

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

color[i]=0;

}

Integer i=0;//用Integer是引用传递

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

//着色方法

*test*(i,adj,color);

i++;

}

//输出着色方案

**for**(**int** j=0;j<color.length;j++){

System.***out***.println(color[j]);

}

}

**public** **static** **void** test(Integer i, **int**[][] adj, **int**[] color){

color[i]++;

**if**(color[i]==5){

//回溯(该节点的候选值都不符合则返回上一个节点)

i=i-1;

*test*(i,adj,color);

}**else**{

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

//如果区域相邻并且颜色相同，那么颜色需要改变

**if**(adj[i][j]==1&&color[i]==color[j]){

*test*(i,adj,color);

}

}

}

方法二：限制上色方法：给a节点上b颜色

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

//测试数据

**int** [][]adj=**new** **int** [][]{

{0,1,0,0,0,0,1},

{1,0,1,1,1,1,1},

{0,1,0,1,0,0,0},

{0,1,1,0,1,0,0},

{0,1,0,1,0,1,1},

{0,1,0,0,1,0,1},

{1,1,0,0,1,1,0}

};

**int** []color=**new** **int**[adj.length];

//初始化color数组，0表示未着色

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

color[i]=0;

}

*getColor*(adj,1,1,1,color,4);

**for**(**int** j=0;j<color.length;j++){

System.***out***.println(color[j]);

}

}

//判断该step节点着色是否符合规则

**static** **boolean** isOk(**int**[][]adj,**int** step,**int** []color){

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

**if**(adj[step-1][i-1]==1&&color[step-1]==color[i-1]){

**return** **false**;

}

}

**return** **true**;

}

**static** **boolean** getColor(**int** [][]adj,**int** step,**int** a,**int** b,**int** []color,**int** colorNum){

//判断是否完成上色

**if**(step>adj.length){

//判断是否是a节点上b颜色

**if**(color[a-1]==b) **return** **true**;

}**else**{

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

color[step-1]=i;

**if**(*isOk*(adj,step,color)){

**if**(*getColor*(adj,step+1,a,b,color,colorNum)) **return** **true**; //回溯法

}**else**{

color[step-1]=0;

}

}

}

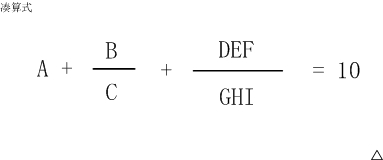
**return** **false**;

}

# 3 dfs题

**dfs**:按照某一顺序遍历，当走不通时，返回到上一个继续按照该顺序走。

## 1 凑算式(dfs)



A-I代表1-9，不同字母代表不同数字，求一共有多少种解法？

**法一(不好理解)：**

**public** **class** javaTest {

**private** **static** **int** *ans* = 0;

**static** **int** []*num*=**new** **int**[10];

**static** **boolean**[]*visit*=**new** **boolean**[10];

**public** **static** **void** dfs(**int** index){

**if**(index == 9)

{

*Solve*();

**return** ;

}

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

{

**if**(!*visit*[i])

{

*visit*[i] = **true**;

*num*[index] = i;

*dfs*(index+1);

*visit*[i] = **false**;

}

}

}

**public** **static** **void** Solve()

{

**double** sum = *num*[0] + (**double**)*num*[1] / *num*[2] + (**double**)(*num*[3]\*100+*num*[4]\*10+*num*[5])/(*num*[6]\*100+*num*[7]\*10+*num*[8]);

**if**(sum == 10)

{

*ans* ++;

}

}

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

*dfs*(0);

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

}

}

**法二(好理解)：**

**public** **class** javaTest {

**static** **int** []*a*=**new** **int**[9];

**static** **int** *ans*=0;

**static** **boolean** judge(**int** []a)//判断是否算式和为10

{

**double** x=a[0]+a[1]\*1.0/a[2]+(a[3]\*100+a[4]\*10+a[5])\*1.0/(a[6]\*100+a[7]\*10+a[8]);

**if**(x==10.0) **return** **true**;

**return** **false**;

}

**static** **boolean** check(**int** num)//检测a[num]是否有重复

{

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

{

**if**(*a*[i]==*a*[num]) **return** **false**;

}

**return** **true**; //没有重复

}

**static** **void** dfs(**int** num)//就是最基本的9个数全排列，在结束条件里用judge()做判断

{

**if**(num>8)

{

**if**(*judge*(*a*))

*ans*++;

**return**;

}

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

{

*a*[num]=i;

**if**(*check*(num))//没有重复

{

*dfs*(num+1);

}

}

}

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

*dfs*(0);

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

}

}

## 2 lake counting

有一个大小为N\*M的园子，雨后积水。八连通的积水被认为是连接在一起的。请求出园子里总共有多少水洼？

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

{

//输入

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

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

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

**char**[][] adj=**new** **char**[n][m];

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

adj[i]=scanner.next().toCharArray();

}

//处理

**int** count=*deal*(adj);

//输出

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

}

**private** **static** **int** deal(**char**[][] adj) {

**int** count=0;

**int** n=adj.length;

**int** m=adj[0].length;

//对值为W的点进行dfs

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

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

**if**(adj[i][j]=='W'){

*dfs*(adj,i,j);

//记录dfs的次数即为所求

count++;

}

}

}

**return** count;

}

**private** **static** **void** dfs(**char**[][] adj, **int** i, **int** j) {

**int** n = adj.length;

**int** m = adj[0].length;

// 标注为访问

adj[i][j] = '\*';

//深度遍历8个方向,方向为左上、上、右上、左、右、左下、下、右下

**for**(**int** a=-1;a<=1;a++){

**for** (**int** b=-1;b<=1;b++){

**int** newX=i+a;

**int** newY=j+b;

//判断新坐标是否满足条件

**if**(newX>=0&&newX<n&&newY>=0&&newY<m&&adj[newX][newY]=='W')

*dfs*(adj,newX,newY);

}

}

//8个方向扫描完毕

**return**;

}