

PROBLEMAS CON CELDAS ADYACENTES

RECORRER CELDAS ADYACENTES A UNA CELDA

Querer recorrer las celdas que están alrededor de una celda dada es una necesidad que surge en multitud de algoritmos de procesamiento de arrays de dos dimensiones. En una matriz, dada un celda `[i][j]`, como mucho puede tener 8 "vecinos" (en rojo)

<code>[i-1][j-1]</code>	<code>[i-1][j]</code>	<code>[i-1][j+1]</code>
<code>[i][j-1]</code>	<code>[i][j]</code>	<code>[i][j+1]</code>
<code>[i+1][j-1]</code>	<code>[i+1][j]</code>	<code>[i+1][j+1]</code>

pero puede tener menos vecinos, por ejemplo ahora la celda `[i][j]` tiene 5 vecinos

<code>[i-1][j]</code>	<code>[i-1][j+1]</code>	
<code>[i][j]</code>	<code>[i][j+1]</code>	
<code>[i+1][j]</code>	<code>[i+1][j+1]</code>	

y ahora la celda `[i][j]` tiene 3 vecinos

	<code>[i][j-1]</code>	<code>[i][j]</code>
	<code>[i+1][j-1]</code>	<code>[i+1][j]</code>

etc.

Si queremos trabajar con las celdas adyacentes, en un método genérico que vale para cualquier `[i][j]` tengo que evitar usar celdas adyacentes fuera de rango que no existen. Vamos a ver esto escribiendo un método genérico sencillo "sumarVecinos".

Ejemplo: `matriz[1][1]` tiene 8 vecinos y no hay problema de indexar fuera de rango

```
class Unidad4{
    static int sumarVecinos(int[][] m, int fila, int col){
        //suponemos que fila y col son índices correctos (en rango)
        return m[fila-1][col-1]+m[fila-1][col]+m[fila-1][col+1]+
            m[fila][col-1]+m[fila][col]+m[fila][col+1]+
            m[fila+1][col-1]+m[fila+1][col]+m[fila+1][col+1];
    }
    public static void main (String[] args){
        int[][] matriz = {
            {2,1,3},
            {4,1,0},
            {1,2,1}
        };
        //sumamos los adyacentes a matriz[1][1], la propia celda no incluida
```

```

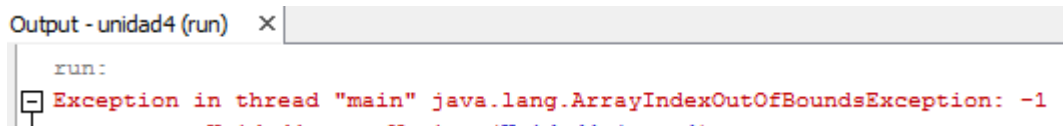
        System.out.println(sumarVecinos(matriz,1,1));
    }
}

```

run:
14

si repito la ejecución con

```
System.out.println(sumarVecinos(matriz,0,0));
```



Output - unidad4 (run) x

```

run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -1

```

Por tanto, el método anterior, antes de sumar una celda, debe de comprobar que está en el rango de la matriz. Vamos a hacer 8 if, cada uno mira si existe uno de los 8 vecinos posibles y si es así suma su valor

```

class Unidad4{
    static int sumarVecinos(int[][] m, int fila, int col){
        //suponemos que fila y col son índices correctos (en rango)
        int total=0;
        //miro de los 8 posibles vecinos, si están en rango en cuyo caso los sumo
        if(fila>0&&col>0)//m[fila-1][col-1] existe y puedo sumarla
            total+=m[fila-1][col-1];
        if(fila>0)//m[fila-1][col] existe y puedo sumarla
            total+=m[fila-1][col];
        if(fila>0&&col<m.length-1)//m[fila-1][col+1] existe y puedo sumarla
            total+=m[fila-1][col+1];
        if(col>0)//m[fila][col-1] existe y puedo sumarla
            total+=m[fila][col-1];
        if(col<m.length-1)//m[fila][col+1] existe y puedo sumarla
            total+=m[fila][col+1];
        if(fila<m.length-1&&col>0)//m[fila+1][col-1] existe y puedo sumarla
            total+=m[fila+1][col-1];
        if(fila<m.length-1)//m[fila+1][col] existe y puedo sumarla
            total+=m[fila+1][col];
        if(fila<m.length-1&&col<m.length-1)//m[fila+1][col+1] existe y puedo sumarla
            total+=m[fila+1][col+1];
        return total;
    }
    public static void main (String[]args){
        int[][] matriz ={
            {2,1,3},
            {4,1,0},
            {1,2,1}
        };
        //sumamos los adyacentes a matriz[1][1], la propia celda no incluida
        System.out.println(sumarVecinos(matriz,0,0));
        System.out.println(sumarVecinos(matriz,2,2));
    }
}

```

La estructura anterior de ifs no lleva else, recuerda que con if-else sólo se ejecutaría una instrucción pero en este caso, una celda puede tener 8 vecinos y querré pasar por los 8 if. Por otro lado, escribir lo anterior, es altamente peligroso porque es muy fácil equivocarse con la lógica de índices, aunque no lo parezca, es más sencillo hacer un for anidado. Para ello razonamos:

- primero proceso *fila-1* , luego *fila* , luego *fila+1*
- dentro de cada fila: primero *col-1*, luego *col*, luego *col +1*

En el siguiente ejemplo recogemos esta idea

```

class Unidad4{
    static int sumarVecinos(int[][] m, int fila, int col){
        //suponemos que fila y col son índices correctos (en rango)
        int total=0;
        for(int f=fila-1;f<fila+2;f++){
            for(int c=col-1;c<col+2;c++){

```

```

        if(f==fila&& c==col){
            continue;
        }
        total+=m[f][c];
    }
}

return total;
}

public static void main (String[]args){
    int[][] matriz ={
        {2,1,3},
        {4,1,0},
        {1,2,1}
    };
    System.out.println(sumarVecinos(matriz,1,1));
    //System.out.println(sumarVecinos(matriz,0,0));
    //System.out.println(sumarVecinos(matriz,2,2));
}
}

```

Con los dos últimos println() indexamos fuera de rango, así que tengo que decidir para cada fila y cada columna si está en rango o no, por ejemplo, en lugar de escribir

```
int f=fila-1
```

para asegurarme que como más bajo voy a indexar la fila 0

```
int f=Math.max(0,fila-1)
```

y en lugar de escribir como límite superior

```
f<fila+2
```

por si acaso fila+1 está fuera de rango escribimos

```
f<Math.min(m.length,fila+2)
```

y un razonamiento análogo para las columnas

```

class Unidad4{
    static int sumarVecinos(int[][] m, int fila, int col){
        //suponemos que fila y col son índices correctos (en rango)
        int total=0;
        for(int f=Math.max(0, fila-1);f<Math.min(m.length, fila+2);f++){
            for(int c=Math.max(0, col-1);c<Math.min(m[f].length, col+2);c++){
                if(f==fila&&c==col){
                    continue;
                }
                total+=m[f][c];
            }
        }

        return total;
    }

    public static void main (String[]args){
        int[][] matriz ={
            {2,1,3},
            {4,1,0},
            {1,2,1}
        };
        System.out.println(sumarVecinos(matriz,1,1));
        System.out.println(sumarVecinos(matriz,0,0));
        System.out.println(sumarVecinos(matriz,2,2));
    }
}

```

En lugar de usar max y min podemos usar el operador condicional para controlar los valores que toman f y c. En este contexto es más claro usar Max y Min, creo yo, de todas formas vemos como quedaría.

```

class Unidad4{
    static int sumarVecinos(int[][] m, int fila, int col){
        //suponemos que fila y col son índices correctos (en rango)

```

```

int total=0;
for(int f=(fila-1<0?0:fila-1);f<(fila+2>m.length?m.length:fila+2);f++){
    for(int c=(col-1<0?0:col-1);c<(col+2>m[f].length?m[f].length:col+2);c++){
        if(f==fila&& c==col){
            continue;
        }
        total+=m[f][c];
    }
}

return total;
}
public static void main (String[] args){
    int[][] matriz = {
        {2,1,3},
        {4,1,0},
        {1,2,1}
    };
    System.out.println(sumarVecinos(matriz,1,1));
    System.out.println(sumarVecinos(matriz,0,0));
    System.out.println(sumarVecinos(matriz,2,2));
}
}

```

otra opción un poco más sucio es utilizar un if que proteja f y c dentro del for anidado

```

static int sumarVecinos(int[][] m, int fila, int col){
    //suponemos que fila y col son índices correctos (en rango)
    int total=0;
    for(int f=fila-1;f<fila+2;f++){
        for(int c=col-1;c<col+2;c++){
            if(f>=0&&f<m.length &&
                c>=0&&c<m[f].length ){
                if(f==fila&&c==col){
                    continue;
                }else{
                    total+=m[f][c];
                }
            }
        }
    }

    return total;
}
}

```

¡Cualquier opción es preferible a la de los 8 ifs!

Ejercicio U4_B4B_E1: Acepta el reto Campo de minas id 176.

Ejercicio U4_B4B_E2: Acepta el reto Sombras en el camping id 207