**SumMatriz**

**Descripción del problema**
La función **SumMatriz** suma dos matrices cuadradas de la misma dimensión, devolviendo una nueva matriz donde cada elemento es la suma de los elementos correspondientes en las matrices originales. Este proceso simula la operación de suma entre matrices en álgebra lineal.

**Funcionamiento de la función SumMatriz**
La función **SumMatriz** fue implementada de la siguiente manera:

```
package taller


class SumMatriz() {
    def sumMatriz(m1: Matriz, m2: Matriz): Matriz = {
        // recibe m1 y m2 matrices cuadradas de la misma dimension , potencia,→ de 2
        val n = m1.length
        require(m1.length == m2.length && m1(0).length == m2(0).length)
        val resultado = Vector.tabulate(n, n)((i, j) => m1(i)(j) + m2(i)(j))
        // y devuelve la matriz resultante de la suma de las 2 matrices
        resultado
    }
}
```

1. **Entrada**: Recibe dos matrices cuadradas m1 y m2, representadas como estructuras de tipo Matriz, que es un Vector[Vector[Int]].
2. **Validación**: Verifica que ambas matrices tienen la misma dimensión, es decir, que el número de filas y columnas coincida.
3. **Operación**: Recorre cada posición (i, j) en las matrices y calcula la suma de los valores correspondientes: m1(i)(j) + m2(i)(j).
4. **Salida**: Devuelve una nueva matriz donde cada posición contiene el resultado de la suma.

**Ejemplo del proceso**
Supongamos que tenemos las siguientes matrices:

- Matriz A:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- Matriz B:

$$\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Cuando aplicamos la función **SumMatriz** a estas matrices, el resultado será:

SumMatriz(A,B)=

$$\begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

```
val obj = new SumMatriz()

val m1 = Vector(
    Vector(1, 2),
    Vector(3, 4)
)
val m2 = Vector(
    Vector(5, 6),
    Vector(7, 8)
)

val result = obj.sumMatriz(m1, m2)

test("SumMatriz Test #1") {
    assert(result(0)(0) == 6)
}

test("SumMatriz Test #2") {
    assert(result(0)(1) == 8)
}

test("SumMatriz Test #3") {
    assert(result(1)(0) == 10)
}

test("SumMatriz Test #4") {
    assert(result(1)(1) == 12)
}

test("SumMatriz Test #5") {
    assert(result.length == m1.length)
    assert(result(0).length == m1(0).length)
}
```

**Casos de prueba**

Se realizaron varios casos de prueba para garantizar que la función **SumMatriz** funcione correctamente en diferentes escenarios. A continuación, se detallan algunos casos representativos:

- **Caso 1.**
  Entrada:

  $$m1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad m2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

  Resultado:

  $$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- **Caso 2.**
  Entrada:

  $$m1 = \begin{bmatrix} -1 & -2 \\ -3 & -4 \end{bmatrix}, \quad m2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

  Resultado:

  $$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

**multMatrizRec**

**Descripción del problema**
La función **multMatrizRec** calcula la multiplicación de dos matrices cuadradas utilizando un enfoque recursivo basado en la técnica de "dividir y conquistar". Divide las matrices en submatrices más pequeñas hasta llegar al caso base, donde realiza el cálculo directamente.

**Funcionamiento de la función multMatrizRec**
La función **multMatrizRec** fue implementada de la siguiente manera:

```scala
package taller

class MultMatriz () {

    def multMatrizRec(m1: Matriz, m2: Matriz): Matriz = {
        val n = m1.length

        if (n == 1) {
            Vector(Vector(m1(0)(0) * m2(0)(0)))
        }

        else {
            val mitad = n/2


            val a11 = SubMatriz(m1, 0, 0, mitad)
            val a12 = SubMatriz(m1, 0, mitad, mitad)
            val a21 = SubMatriz(m1, mitad, 0, mitad)
            val a22 = SubMatriz(m1, mitad, mitad, mitad)

            val b11 = SubMatriz(m2, 0, 0, mitad)
            val b12 = SubMatriz(m2, 0, mitad, mitad)
            val b21 = SubMatriz(m2, mitad, 0, mitad)
            val b22 = SubMatriz(m2, mitad, mitad, mitad)

            val p1 = SumMatriz(multMatrizRec(a11, b11), multMatrizRec(a12, b21))
            val p2 = SumMatriz(multMatrizRec(a11, b12), multMatrizRec(a12, b22))
            val p3 = SumMatriz(multMatrizRec(a21, b11), multMatrizRec(a22, b21))
            val p4 = SumMatriz(multMatrizRec(a21, b12), multMatrizRec(a22, b22))

            Vector.tabulate(n, n)((i, j) =>
                if (i < mitad && j < mitad) p1(i)(j)
                else if (i < mitad && j >= mitad) p2(i)(j - mitad)
                else if (i >= mitad && j < mitad) p3(i - mitad)(j)
                else p4(i - mitad)(j - mitad)
            )
        }
    }
}
```

1. **Entrada**: Recibe dos matrices cuadradas m1 y m2 del tipo Matriz.
2. **Caso base**: Si la dimensión de las matrices es 1×1, multiplica directamente los elementos de ambas matrices.
3. **División**: Divide cada matriz en cuatro submatrices iguales:

- a11,a12,a21,a22 para m1.
- b11,b12,b21,b22 para m2.

4. **Recursión**: Calcula las submatrices del resultado usando combinaciones de las submatrices:

- $p1 = \text{multMatrizRec}(a_{11}, b_{11}) + \text{multMatrizRec}(a_{12}, b_{21})$
- $p2 = \text{multMatrizRec}(a_{11}, b_{12}) + \text{multMatrizRec}(a_{12}, b_{22})$
- $p3 = \text{multMatrizRec}(a_{21}, b_{11}) + \text{multMatrizRec}(a_{22}, b_{21})$
- $p4 = \text{multMatrizRec}(a_{21}, b_{12}) + \text{multMatrizRec}(a_{22}, b_{22})$

5. **Composición**: Combina las submatrices p1,p2,p3,p4 en una matriz resultante completa.
6. **Salida**: Retorna la matriz resultante.

**Ejemplo del proceso**
Supongamos que tenemos las siguientes matrices:

- Matriz A:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- Matriz B:

$$\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Cuando aplicamos la función **multMatrizRec**, obtenemos:

1. División en submatrices:
   - A11=[1], A12=[2],…
2. Cálculo recursivo de p1,p2,p3,p4.
3. Combinar submatrices para formar el resultado:

   multMatrizRec(A,B)=

$$\text{multMatrizRec}(A, B) = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

**Casos de prueba**

```
test("Multiplicación de matrices 1x1") {
    val m1 = Vector(Vector(2))
    val m2 = Vector(Vector(3))
    val expected = Vector(Vector(6))
    assert(multMatriz.multMatrizRec(m1, m2) == expected)
}

test("Multiplicación de matrices 2x2") {
    val m1 = Vector(
        Vector(1, 2),
        Vector(3, 4)
    )
    val m2 = Vector(
        Vector(5, 6),
        Vector(7, 8)
    )
    val expected = Vector(
        Vector(19, 22),
        Vector(43, 50)
    )
    assert(multMatriz.multMatrizRec(m1, m2) == expected)
}
```