Programação Estruturada

Aula 14 - Recursão: Arrays, Matrizes e Exercícios

Videoaula 05: Comparando e Somando Matrizes

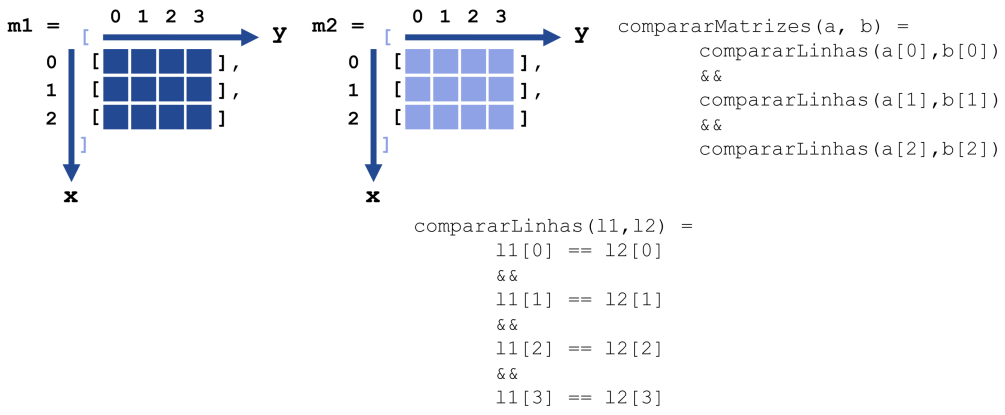
Videoaula 05: Comparando e Somando Matrizes

Nesse instante, continuarei com um outro exemplo da aplicação de recursão em matrizes. O próximo apresentará uma função recursiva que indica se duas matrizes são iguais. Para isso, será necessário que ambas tenham as mesmas dimensões e que todos os seus elementos sejam iguais.

Aqui, utilizarei a estratégia de "dividir para conquistar". Primeiro, definimos uma função que compara dois arrays. Logo após, usando essa função, irei comparar todas as linhas das matrizes entre si. Obviamente, as matrizes só serão iguais se todas as suas linhas forem iguais.

Para exemplificar, veja como farei a comparação das das duas matrizes do slide (Figura 1). A matriz `m1` (azul) e a matriz `m2` (azul-claro).

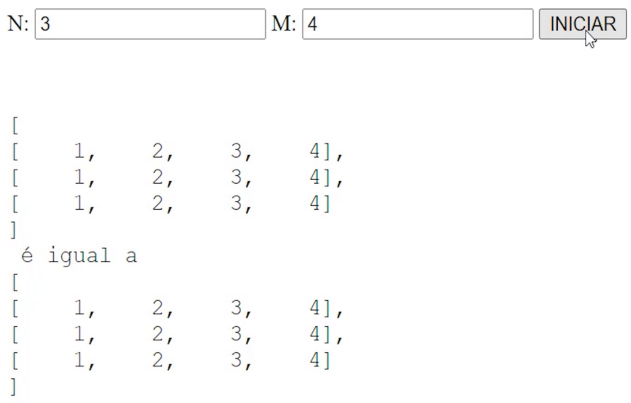
Figura 1 - Comparando Duas Matrizes



Para isso, é necessário realizar as comparações das linhas `m1[0]` com `m2[0]`, das linhas `m1[1]` com `m2[1]` e das linhas `m1[2]` com `m2[2]`. Todas as três comparações devem ser verdadeiras para que as duas matrizes sejam iguais.

Primeiro, veja como ficaria a definição recursiva da função `compararLinhas`, a qual implementa essa comparação de linhas. Nessa função, terei que comparar os elementos correspondentes. Por exemplo, para comparar a primeira linha das duas matrizes, terei que comparar os elementos correspondentes. Nesse exemplo, considerando as linhas `l1` e `l2`, terei a comparação de `l1[0]` com `l2[0]`, de `l1[1]` com `l2[1]`, de `l1[2]` com `l2[2]` e de `l1[3]` com `l2[3]`.

Figura 2 - Comparando Linhas



Veja como fica a implementação recursiva dessa função.

Nesse exemplo, temos uma função (Linha 41) que compara duas linhas, `a` e `b`, e dois casos base (Linhas 45 e 48). Basicamente, se eles tiverem tamanhos diferentes, essas linhas serão diferentes porque têm tamanhos diferentes.

Código 1 - 14_8 Compara Matrizes.html

```
1  <html >
2    <head>
3      <meta charset="UTF-8" />
4      <title>Programação Estruturada - Aula 14</title>
5    </head>
6    <style>
7      pre {
8        font-family: "Courier New", Courier, monospace;
9      }
10   </style>
11   <body>
12     <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
```

```
13
14      <h1>Compara Matrizes</h1>
15
16      N: <input type="number" id="number_N" size="3" value="">
17      M: <input type="number" id="number_M" size="3" value="">
18      <button id="btn_iniciar" onclick="iniciar()">INICIAR</button>
19      <pre>
20          <p id="resposta"></p>
21      </pre>
22
23      <script src="script.js"></script>
24    </body>
25 </html>
26
```

```
1  var n = 0;
2  var m = 0;
3
4  function iniciar() {
5     n = Number(document.getElementById("number_N").value);
6     m = Number(document.getElementById("number_M").value);
7
8     var resposta = "";
9     if (n > 0 && m > 0) {
10       var matrizA = inserir();
11       var matrizB = inserir();
12       var igual = compararMatrizes(matrizA, matrizB);
13       resposta = exibirMatriz(matrizA) + "<br> ";
14       if (!igual) {
15          resposta = resposta + "NÃO "
16       }
17       resposta = resposta + "é igual a <br>"
18       resposta = resposta + exibirMatriz(matrizB);
19    } else {
20       resposta = "Favor inserir valores positivos para as dimensões N e M."
21    }
22    document.getElementById("resposta").innerHTML = resposta;
23 }
24
25 function compararMatrizes(a,b) {
26    var resposta = true;
27    // Caso base, as matrizes têm tamanhos diferentes
28    if (a.length != b.length) {
29       resposta = false;
30    // Caso base, chegamos ao final das matrizes
31    } else if (a.length == 0) {
32       resposta = true;
33    // Caso recursivo
```

```
    } else {
        resposta = compararLinhas(a[0],b[0]);
        resposta = resposta && compararMatrizes(a.slice(1), b.slice(1));
    }
    return resposta;
}

function compararLinhas(a,b) {
    var resposta = true;

    // Caso base, os arrays têm tamanhos diferentes
    if (a.length != b.length) {
        resposta = false;
    // Caso base, chegamos ao fim do array
    } else if (a.length == 0) {
        resposta = true;
    // Caso recursivo
    } else {
        resposta = (a[0] == b[0]);
        resposta = resposta && compararLinhas(a.slice(1), b.slice(1));
    }
    return resposta;
}

// Função para Inserir Matriz
function inserir() {
    var matriz = [];
    for (var i = 0; i < n; i++) {
        var linha = [];
        for (var j = 0; j < m; j++) {
            do {
                num_str = prompt("Digite o elemento "+i+" x "+j+":");
                num = Number(num_str);
                if (!invalido(num)) {
                    linha.push(num);
                } else {
                    alert("Valores devem ser números entre -999 e 1000")
                }
            } while (invalido(num));
        }
        matriz.push(linha);
    }
    return matriz;
}

function invalido(num){
    return isNaN(num) || num < -999 || num > 1000;
}
```

```
82
83 // Função para Exibir Matriz
84 function exibirMatriz(matriz){
85    var resposta = "[<br>";
86    for (var i = 0; i < matriz.length; i++) {
87       resposta = resposta + exibirLinha(matriz[i]);
88       if (i + 1 < matriz.length) {
89          resposta = resposta + ", <br>";
90       }
91    }
92    resposta = resposta + "<br>]";
93    return resposta;
94 }
95
96 function exibirLinha(linha){
97    var resposta = "[";
98    for (var i = 0; i < linha.length; i++) {
99       resposta = resposta + exibirElemento(linha[i]);
100      if (i + 1 < linha.length) {
101         resposta = resposta +",";
102      }
103   }
104   resposta = resposta + "]";
105   return resposta;
106 }
107
108 function exibirElemento(elemento){
109   var tam = elemento.toString().length;
110   var brancos = 5 - tam;
111   var resposta = "";
112   for(var i=0; i < brancos; i++) {
113      resposta = resposta + " ";
114   }
115   resposta = resposta + elemento;
116   return resposta;
117 }
118
```

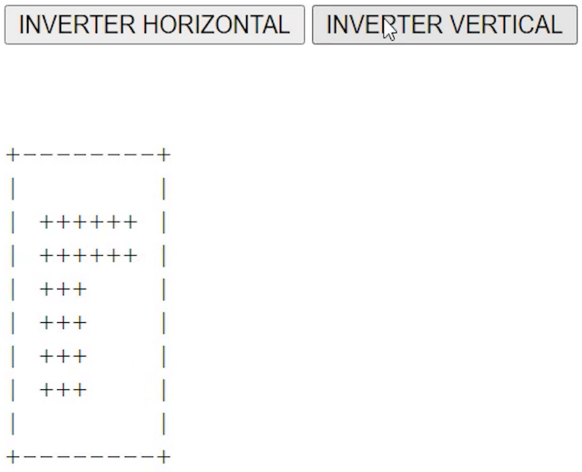
Então, se o `a.length` for diferente de `b.length`, a resposta é `false` (linha 46), tá certo? Caso contrário, sabemos que as duas linhas, os dois arrays, têm o mesmo tamanho. Dessa forma, eles estão ok e vão chegar no fim ao mesmo tempo.

Se o `a.length` for igual a 0 é porque chegamos ao fim de ambos, assim teremos `true` (linha 49). E o caso recursivo? Para o caso recursivo, temos a `resposta = (a[0] == b[0])`, ou seja, temos que os dois primeiros elementos, o `a[0]` é igual a `b[0]` (Linha 52) e a comparação do resto também é `true` (linha 53), tá certo?

Na linha 53 temos a `resposta` e `compararLinhas` do resto da calda do array `a`, que é o `a.slice(1)`, com o resto da calda do array `b`, que é o `b.slice(1)`, e também tem que ser `true`, tá certo? Então, para todo elemento teremos o caso em que a resposta, ou seja, os dois primeiros elementos são iguais e a comparação do resto também é igual, assim ele retorna `true`, ok?

Agora, usando essa função, posso finalmente comparar duas matrizes. O raciocínio da função `compararMatrizes` é muito parecido com o que acabamos de ver para a função `compararLinhas`. A única diferença é que agora os elementos são arrays. Portanto, para comparar, por exemplo, o elemento `a[0]` da matriz `a` com o elemento `b[0]` da matriz `b`, usei a função de comparação de linhas e não simplesmente `==`.

Figura 3 - Comparando Matrizes



A função `compararLinhas` será utilizada para comparar todas as linhas da matriz. Veja como ficará a implementação da função `compararMatrizes` e como posso encaixá-la em uma solução final.

Como agora eu tenho a comparação de linhas, posso comparar as matrizes. É muito parecido com a comparação de linhas, mas é óbvio que precisaremos considerar que estamos comparando matrizes que têm linhas dentro dela, tá?

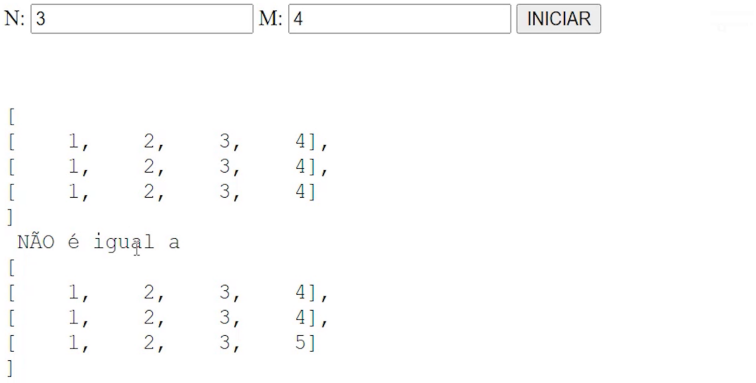
Os casos base, por exemplo, são exatamente os mesmos, ou seja, se duas matrizes têm arrays com tamanhos diferentes, é porque elas têm dimensões diferentes, então elas são diferentes, tá? Então se o `a.length` for diferente de `b.length`, a resposta é `false` (linhas 28 e 29). Agora, se tiverem o mesmo tamanho, passa pela outra condição da linha 31, que é o outro caso base, pois terminamos ambas as matrizes ao mesmo tempo, e quando isso acontece a resposta é `true` e, finalmente, temos o caso recursivo (linha 34).

No caso recursivo, teremos que comparar os dois primeiros elementos, e ambos são arrays, são linhas, então chamamos a função `compararLinhas(a[0],b[0])` e não dizemos simplesmente que um é igual ao outro, usamos a função `compararLinhas`, que acabamos de conhecer.

Depois, devemos verificar que a comparação do resto da matriz também é `true`, tá? Comparamos o resto da matriz `a` com `a.slice(1)` com o resto da matriz `b`, que é o `b.slice(1)`. Então, novamente, fazemos a comparação das linhas, e verificamos que as duas linhas são iguais e, veja o conector lógico na linha 36, ele compara as linhas e a comparação do resto das matrizes também é igual.

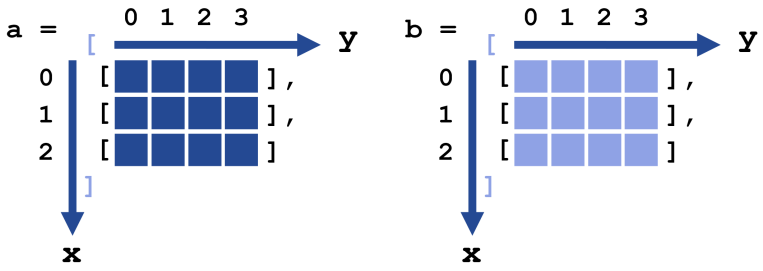
Agora, vejamos como encaixamos isso numa solução completa. Vou primeiro mostrar a página HTML que criei para usar esse exemplo. Primeiro, você define qual a dimensão `n` e `m` das duas matrizes e depois insere todos os elementos dessas matrizes. Para isso, vamos dizer que temos a matriz 3 por 4, e ao iniciar ele vai perguntar quem é o elemento 0 x 0 da matriz, vou dizer que é 1, 0 x 1 é 2, 0 x 2 é 3, 0 x 3 é 4. Agora, a gente tem que inserir a outra matriz. Vamos inserir uma matriz igual [[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]. Em seguida, ele vai dar resposta usando aquela comparação para conferir se as matrizes são iguais.

Figura 4 - Compara Matrizes



Se quisermos fazer uma versão diferente, ou seja, mudar só um elemento, acrescento o 5 e agora, porque eu mudei um elemento, ele está informando que as matrizes não são iguais.

Figura 5 - Resultado para Matrizes Diferentes



Como fiz isso? Vamos ver primeiro o HTML (Código 1), nele temos dois campos, `n` e `m`, e o botão iniciar, que chama a função `iniciar`, ou seja, é simples. Então, veremos agora como está a comparação de matrizes no nosso JavaScript.

Eu tenho meu `n` e `m`, que serão as minhas dimensões. Eu pego essas duas dimensões, e a primeira coisa que eu faço é verificar que elas realmente são maiores que 0, então não posso passar, por exemplo, um valor -1; se eu fizer isso, ele indicará uma mensagem de erro dizendo: `Favor inserir valores positivos para as`

`dimensões N e M`. Na linha 20, tenho essa mensagem de erro, mas fiz a verificação de que os dois são positivos. Então, chamo a função `inserir` (linhas 10 e 11), já vimos como ela funciona, para inserção dos elementos das matrizes.

Para verificar minha matriz, faço um laço `for` (linha 61), com `i` variando de 1 até antes de `n` e para cada um, construo uma linha usando também outro laço `for` (linha 63). Na linha 65, informo qual é o `i` e o `j`. Pego esse valor do *prompt* (linha 66) e transformo em número; se ele não for inválido, coloco para dentro, caso contrário, eu digo que o valor deve ser entre 999 e 1000 (linha 70). Coloquei essa restrição para melhorar a exibição das matrizes. E vou fazer isso enquanto ele for inválido. Quando terminar esse laço (linhas 63-73), terminei de pegar a linha, então faço um `push` daquela linha que acabei de construir.

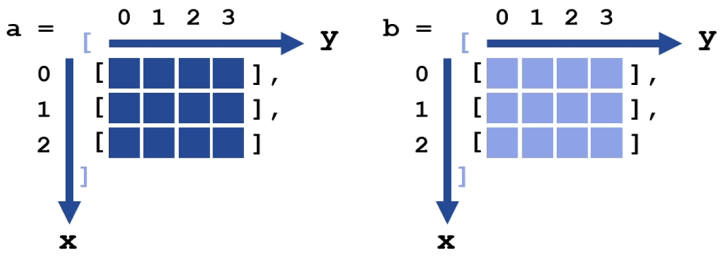
O que é uma entrada inválida? Ela é inválida se ela não for um número, nesse caso, se ela for menor que 999 ou se for maior que 1000, ok? Então, termino de inserir a matriz `a` e a matriz `b`, e chamo a função `compararMatrizes`. Essa função vai ver se uma matriz é igual a outra, e retornará `true` ou `false` (linha 12). Então, para a resposta temos primeiro a exibição da primeira matriz. Se a resposta for não igual, ou seja, se a comparação deu negativa (linha 14), vou colocar a palavra "NÃO" (linha 15); caso contrário, não coloco nada e vai ficar o texto "é igual a" seguido da exibição da outra matriz, tá certo? Com isso, a gente consegue encaixar a comparação das matrizes nessa solução mais completa que acabei de mostrar.

A estratégia de "dividir para conquistar" pode também ser usada em diversas outras soluções, não só para problemas envolvendo matrizes, mas várias outras estruturas de dados como, por exemplo, árvores e grafos. Você, com certeza, fará essa observação à medida que for conhecendo essas outras estruturas de dados ao longo de sua formação.

Para apresentar um exemplo final, lembra-se do uso de recursão para somar os elementos correspondentes de dois arrays? Se não, reveja a **Videoaula 01: Recursão e Arrays**.

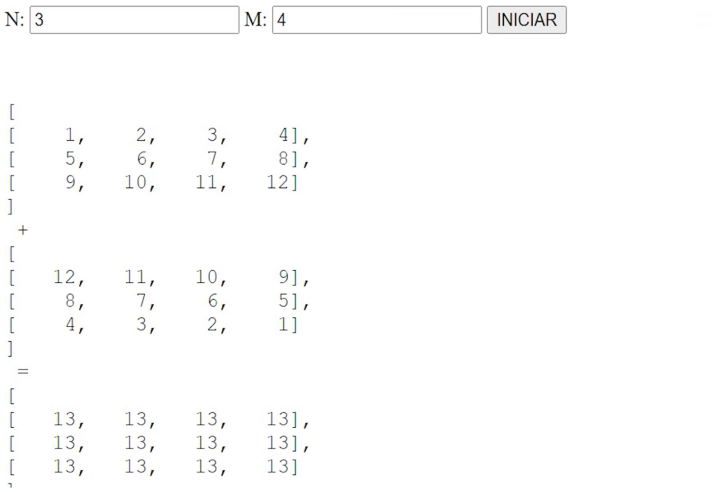
Podemos utilizar essa solução para somar os elementos correspondentes de duas matrizes. Isto porque, de maneira muito similar ao último exemplo, a soma das linhas das matrizes pode ser feita usando a função específica de soma de arrays já vista por você. Por exemplo, para somar a primeira linha das matrizes, pode-se usar a função que soma os arrays fazendo simplesmente uma soma dos elementos correspondentes.

Figura 6 - Somando Linhas



A soma das matrizes será um array cujos elementos são os resultados dessas somas.

Figura 7 - Somando Matrizes



Veja como fica a solução final.

Nesse exemplo, iremos inserir as duas matrizes usando a mesma maneira que vimos no exemplo anterior. E aí, após essa inserção, ele vai retornar a matriz que é o resultado da soma dessas duas matrizes que nós inserirmos, tá certo?

Código 2 - 14_9 Soma Matrizes.html

```html
<html >
  <head>
    <meta charset="UTF-8" />
    <title>Programação Estruturada - Aula 14</title>
  </head>
  <style>
    pre {
      font-family: "Courier New", Courier, monospace;
    }
  </style>
  <body>
    <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

    <h1>Soma Matrizes</h1>

    N: <input type="number" id="number_N" size="3" value="">
    M: <input type="number" id="number_M" size="3" value="">
    <button id="btn_iniciar" onclick="iniciar()">INICIAR</button>
    <pre>
      <p id="resposta"></p>
    </pre>

    <script src="script.js"></script>
  </body>
</html>

```

```
var n = 0;
var m = 0;

function iniciar() {
  n = Number(document.getElementById("number_N").value);
  m = Number(document.getElementById("number_M").value);

  var resposta = "";
  if (n > 0 && m > 0) {
    var matrizA = inserir();
    var matrizB = inserir();
    var mesmaDimensoes = compararDimensoes(matrizA, matrizB);
```

```
13
14      if (!mesmaDimensoes) {
15          resposta = "Favor inserir matrizes com as mesmas dimensões."
16      } else {
17          var matrizSoma = somarMatrizes(matrizA,matrizB);
18          resposta = exibirMatriz(matrizA) + "<br> + <br>";
19          resposta = resposta + exibirMatriz(matrizB) + "<br> = <br>";
20          resposta = resposta + exibirMatriz(matrizSoma);
21      }
22    } else {
23      resposta = "Favor inserir valores positivos para as dimensões N e M."
24    }
25    document.getElementById("resposta").innerHTML = resposta;
26  }
27
28  // Esta função assume que as matrizes têm as mesmas dimensões.
29  function somarMatrizes(m1,m2) {
30    var resposta;
31    if (m1.length == 0) {
32      resposta = [];
33    } else {
34      resposta = [somarArrays(m1[0],m2[0])];
35      resposta = resposta.concat(somarMatrizes(m1.slice(1), m2.slice(1)));
36    }
37    return resposta;
38  }
39
40  // Soma de Arrays
41  function somarArrays(l1, l2) {
42    var soma;
43    if (l1.length == 0) {
44      soma = l2;
45    } else if (l2.length == 0) {
46      soma = l1;
47    } else {
48      var somaDoResto = somarArrays(l1.slice(1), l2.slice(1));
49      soma = [l1[0] + l2[0]];
50      soma = soma.concat(somaDoResto);
51    }
52    return soma;
53  }
54
55  // Comparar Dimensões
56  function compararDimensoes(a,b) {
57    var resposta = true;
58    if (a.length != b.length) {
59      resposta = false;
60    } else if (a.length == 0) {
```

```
        resposta = true;
    } else {
        resposta = (a[0].length == b[0].length);
        resposta = resposta && compararDimensoes(a.slice(1), b.slice(1));
    }
    return resposta;
}

// Função para Inserir Matriz
function inserir() {
    var matriz = [];
    for (var i = 0; i < n; i++) {
        var linha = [];
        for (var j = 0; j < m; j++) {
            do {
                num_str = prompt("Digite o elemento "+i+" x "+j+":");
                num = Number(num_str);
                if (!invalido(num)) {
                    linha.push(num);
                } else {
                    alert("Valores devem ser números entre -999 e 1000")
                }
            } while (invalido(num));
        }
        matriz.push(linha);
    }
    return matriz;
}

function invalido(num){
    return isNaN(num) || num < -999 || num > 1000;
}

// Função para Exibir Matriz
function exibirMatriz(matriz){
    var resposta = "[<br>";
    for (var i = 0; i < matriz.length; i++) {
        resposta = resposta + exibirLinha(matriz[i]);
        if (i + 1 < matriz.length) {
            resposta = resposta + ", <br>";
        }
    }
    resposta = resposta + "<br>]";
    return resposta;
}

function exibirLinha(linha){
    var resposta = "[";
```

```
109     for (var i = 0; i < linha.length; i++) {
110         resposta = resposta + exibirElemento(linha[i]);
111         if (i + 1 < linha.length) {
112             resposta = resposta +",";
113         }
114     }
115     resposta = resposta + "]";
116     return resposta;
117 }
118 
119 function exibirElemento(elemento){
120     var tam = elemento.toString().length;
121     var brancos = 5 - tam;
122     var resposta = "";
123     for(var i=0; i < brancos; i++) {
124         resposta = resposta + " ";
125     }
126     resposta = resposta + elemento;
127     return resposta;
128 }
129 
```

Então, o nosso HTML é exatamente o mesmo que vimos anteriormente. Mas aqui a função `iniciar` é um pouco diferente. Ela pega também as duas dimensões, a dimensão `n` e a dimensão `m` (linhas 5 e 6). Chama novamente a função `inserir` para criar a matriz `a` e a matriz `b`. Só que a primeira coisa que a gente fez foi, apesar da nossa inserção não permitir que insira matrizes de dimensões diferentes (linhas 10 e 11), colocar, por redundância, na linha 12, para que isso possa ser usado em outro contexto, a função `compararDimensoes`, ou seja, comparar que as duas matrizes têm as mesmas dimensões (Linha 56).

Então, começamos com a resposta `true`, e se as duas matrizes têm a quantidade de linhas diferentes, ou seja, `a.length` é diferente de `b.length`, a resposta é `false`, então esse é o caso base.

Outro caso base é quando chegamos no fim de ambas as matrizes, ou seja, quando `a.length` for igual a 0, nesse caso a resposta é `true`. Caso contrário, temos o caso recursivo. Ou seja, comparamos as duas primeiras linhas, elas têm que ter o mesmo tamanho, e a comparação do resto também tem que ser `true`, tá certo?

Então, dizemos que a `resposta` é a comparação do tamanho de `a[0]` com o tamanho de `b[0]`. Assim, `a[0].length` tem que ser igual a `b[0].length`, ou seja, as duas linhas têm o mesmo tamanho (linha 63).

E a comparação das dimensões dos restos são iguais, ou seja, a comparação de `a.slice(1)` com `b.slice(1)` usando `compararDimensoes` é igual, tá certo? Se ele não passou por esse teste, teremos a mensagem de erro. No nosso exemplo isso não vai acontecer porque a gente não consegue inserir matrizes de tamanhos diferentes, mas se isso fosse possível, ele ia dá a mensagem de erro: “Favor inserir matrizes com as mesmas dimensões.” (linha 15).

Então, chamamos a função `somarMatrizes`, passando a matriz `a` e a matriz `b`, será exibida a matriz `a`, o sinal de soma, a matriz `b`, o sinal de igual, e a matriz `soma` que a gente construiu (linhas 18 a 20).

Logo, nosso foco nesse exemplo é na função `somarMatrizes`, mas como podemos somar matrizes? Lembrando que já temos a garantia que elas têm a mesma dimensão. Assim, se eu cheguei ao final da primeira, acabei, passei da última linha e vou retornar o array vazio. Caso contrário, vou colocar como primeiro elemento a soma dos arrays das duas primeiras linhas (Linha 34). Então, vou pegar `m1[0]` e `m2[0]` e vou somar essas duas linhas para ter uma linha que será o primeiro elemento desse array.

Na minha variável `resposta`, vou concatenar isso com o resto das somas (linha 35), tá certo? Com a matriz resultante das somas, ou seja, `somarMatrizes(m1.slice(1),m2.slice(1))`.

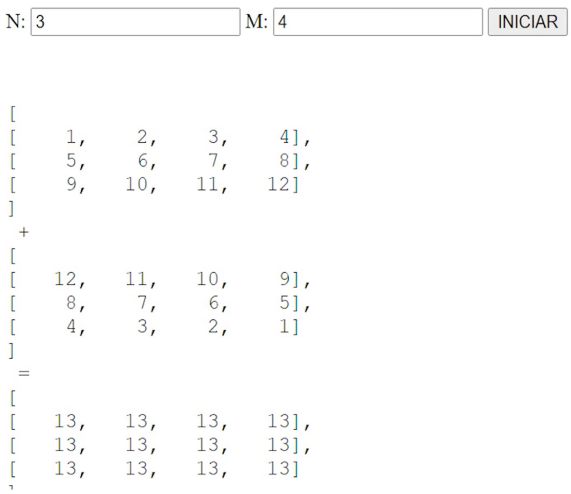
Por fim, ficou faltando apenas conhecer como somo dois arrays. Para isso, utilizamos a função `somarArrays` (Linha 41), já vista anteriormente. Temos dois arrays, `l1` e `l2`. Se o `l1.length==0`, a gente tem o `l2`, e se `l2.length==0`, a gente tem o `l1`.

Note que essa é a versão genérica da soma de arrays que já conhecemos, onde os dois arrays podem ter tamanhos diferentes, mas nesse caso, obviamente, os dois arrays não terão tamanhos diferentes, tá certo?

A soma do resto é feita a partir da soma dos arrays (linha 48), que é o `slice` do primeiro, a calda do primeiro com a calda do segundo, então temos a soma dos primeiros elementos (Linha 49) concatenada com a soma do resto (linha 50).

Com isso, temos a soma das linhas dentro do contexto da soma das matrizes. Assim, na página HTML, vamos pegar, por exemplo, a matriz 3 por 4, e vamos começar o processo inserindo números de 1 a 12. Passamos para a próxima matriz e vamos inserir números de 12 a 1. Logo, temos a soma das matrizes resultando 13 em todos os elementos, porque 1 mais 12 é 13, 2 mais 11 é 13, e assim por diante, ok?

Figura 8 - Soma de Matrizes



Chegamos ao final e, mais uma vez, peço a você que não deixe de praticar a fim de absorver muito bem esse conteúdo. Para isso, deixarei uma lista de exercícios. **Lembre-se**: recursão é uma ferramenta importantíssima na programação! Tchau, tchau!