TwoD\_fun

In this problem, you will implement three static methods from the TwoD\_fun class.

The first static method, getLargest(int a, int b, int c,int d), which given four nonnegative numbers, arranges them to form the largest integer possible. The four parameters will all be greater than or equal to 0, and the values may repeat. You may assume the return value will be a valid int value.

The following code shows the results of the getLargest method.

|  |  |
| --- | --- |
| The following code | Returns |
| TwoD\_fun td = new TwoD\_fun(); |  |
| td.getLargest(1, 5, 2, 3); | 5321 |
| td.getLargest(13, 21, 8, 15); | 8211513 |
| td.getLargest(60, 52, 79, 8); | 8796052 |
| td.getLargest(8, 6, 9, 12); | 98612 |
| td.getLargest(1, 20, 2, 10); | 220110 |
| td.getLargest(1, 10, 13, 130); | 13130110 |

The second static method is findLargest(int[][] arr)). This method, given a 4 x 4 array, returns the largest integer that can be formed by rearranging the values in any row or any column. You may assume all values are greater than or equal to 0 and the numbers may repeat. You may also assume the return value will be a valid int value.

The following code shows the results of the findLargest method.

|  |  |
| --- | --- |
| The following code | Returns |
| TwoD\_fun td = new TwoD\_fun(); |  |
| int[][] arr1 = {{1, 2, 3, 4}, { 2, 3, 4, 5},  {3, 4, 5, 6}, {4, 5, 6, 7} }; |  |
| ans = td.findLargest(arr1); // column 4 | 7654 |
| int[][] arr2 = {{16, 9, 3, 14},  { 24, 38, 7, 8}, {6, 42, 55, 0}, {4, 5, 29, 7} }; |  |
| td.findLargest(arr2) // column 2 | 954238 |
| int[][] arr3 = {{9, 63, 25, 14}, { 24, 38, 7, 8},  {63, 42, 55, 0}, {14, 5, 29, 7} }; |  |
| td.findLargest(arr3) | 9632514 |

The third static method is biggestSquare(int[][] arr). This method, given an int[][] containing only 0’s and 1’s, and returns the area of largest square made of all 1's. You may assume:

* arr.length == arr[0].length
* arr[k].length == arr[m].length, 0 <= k, m < arr.length
* arr[k] == 0 || arr[k] == 1, 0 <= k, m < arr.length

The following code shows the results of the biggestSquare method with m == 1.

|  |  |
| --- | --- |
| The following code | Returns |
| TwoD\_fun td = new TwoD\_fun(); |  |
| int[][] arr1b = {{**1, 1, 1**},  {**1, 1, 1**},  {**1, 1, 1**} }; |  |
| td.biggestSquare(arr1b); | 9 |
| int[][] arr2b = {{**1, 1, 1, 1**, 1},  {**1, 1, 1, 1**, 1},  {**1, 1, 1, 1**, 1},  {**1, 1, 1, 1**, 1},  {1, 1, 1, 1, 0} }; |  |
| td.biggestSquare(arr2b); | 16 |
| int[][] arr3b = {{1, 0, 1, 1, 1},  {1, **1, 1, 1**, 1},  {1, **1, 1, 1**, 0},  {0, **1, 1, 1**, 1},  {1, 1, 1, 1, 1} }; |  |
| td.biggestSquare(arr3b); | 9 |
| int[][] arr4b = {{1, 1, 1, 0, 1},  {1, 0, 1, 1, 1},  {1, 1, 1, 0, 1},  {1, 0, 1, **1, 1**},  {1, 1, 0, **1, 1**} }; |  |
| td.biggestSquare(arr4b); | 4 |