Version 1

Q1.There is a set of N jars containing chocolates. Some of them may be empty. Determine the maximum number of chocolates Andrew can pick from the jars given that he cannot pick jars next to each other

最大巧克力数:类比House Robberhttps://leetcode.com/problems/house-robber/

input: int number of jars, int[] representing the number of chocolate

output: int max

example:

input: 6, [5 30 99 60 5 10]

output: 114

取5 99 10

**

```
public static int rob(int[] nums) {
        // corner case
        if (nums == null || nums.length == 0) return 0;
        if (nums.length == 1) return nums[0];
        // general case
        int[] dp = new int[nums.length];
        // base case
        dp[0] = nums[0];
        dp[1] = Math.max(nums[1], dp[0]);
        for (int i = 2; i < nums.length; i++) {
            dp[i] = Math.max(dp[i - 1], nums[i] + dp[i - 2]);
        }
        return dp[nums.length - 1];
    }
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int length = scanner.nextInt();
    int[] nums = new int[length];
    for (int i = 0; i < length; i++) {
        nums[i] = scanner.nextInt();
    }
    int result = rob(nums);
    System.out.println(result);
}
```

Q2. Ray likes puzzles. One day, he challenged Ansh with a puzzle to find a string that is the same when read forwards and backwards

找最长回文substring，如有相同取字母最小：类比 https://leetcode.com/problems/longest-palindromic-substring/

input: String input

output: String substring palindrome

example:

    input: YABCCBAZ

    output: ABCCBA

** implementation

```
public static void longestPalindrome(String input) {

    // corner case

    if (input == null || input.length() == 0) {

        System.out.printLn(“None”);

        return;

    }
```

```
      // general case

      int maxLength = 1;

      String maxSubstring = input.substring(0, 1);

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

            String substring = findPalindrome(input, i);

            if (substring.length() > maxSubstring) {

                  // update if we find a longer substring

                  maxSubstring = substring;

            } else if (substring.length() == maxSubstring.length() &&
      substring.compareTo(maxSubstring < 0)) {

                  // if have the same length, choose lexicographically
      smallest one

                  maxSubstring = substring;

            }

      }

      // check update or not

      if (maxSubstring.length() == 1) {

            System.out.println(“None”);

            return;

      }

      System.out.println(maxSubstring);

}

// helper function to check palindrome
```

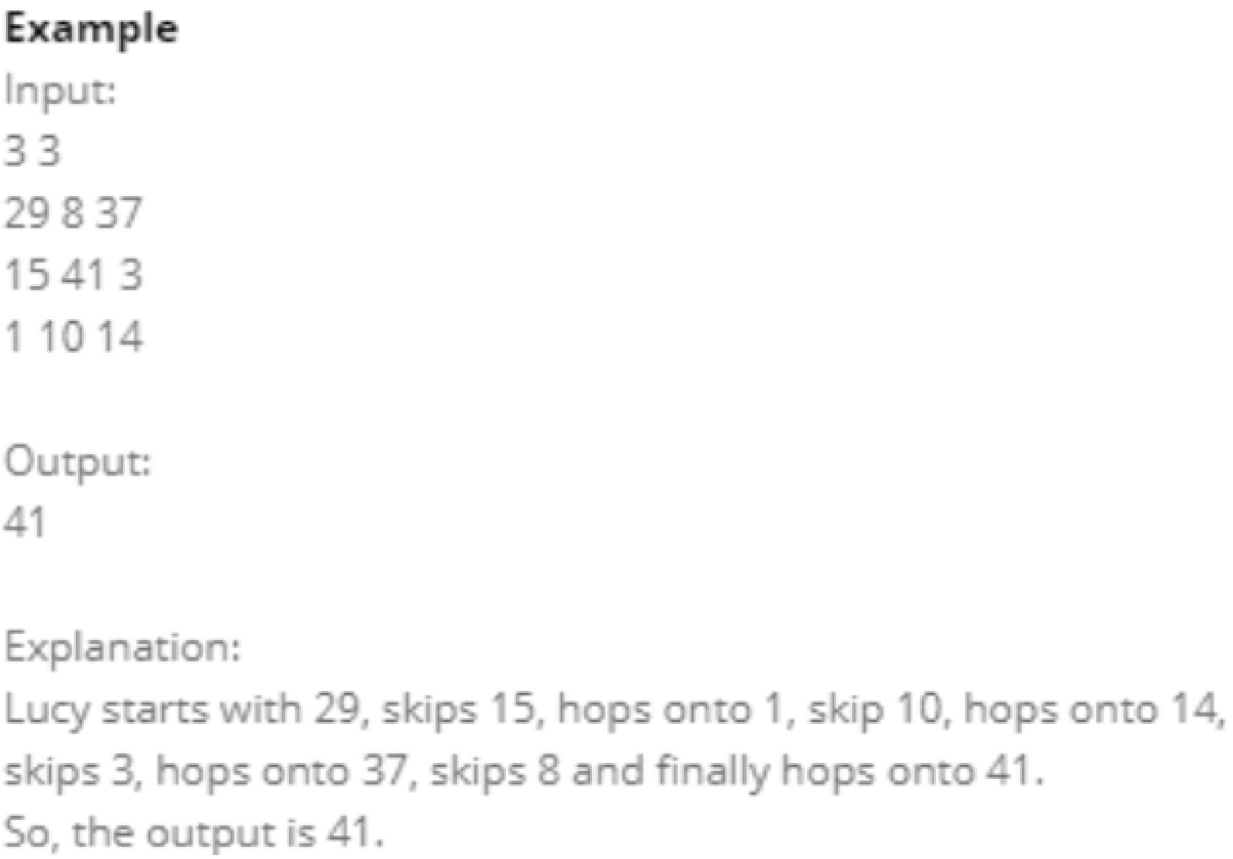
```
private static String findPalindrome(String s, int i) {

      int j = i - 1, k = i + 1;

      String result = “”;

      // start with i in the middle

      while (j >= 0 && k < s.length() && s.charAt(j) == s.charAt(k)) {

            j–;

            k++;

      }

      result = s.substring(j + 1, k);

      // start with i and i - 1 in the middle

      j = i - 1;

      k = i;

while (j >= 0 && k < s.length() && s.charAt(j) == s.charAt(k)) {

            j–;

            k++;

      }

      // compare two cases

      if (k - j + 1 > result.length()) return s.substring(j+ 1, k);

      return result;

}
```

Q3. Lucy loves to play the Hop, skip and jump game. Given an N*M matrix and starting from cell (1,1), her challenge is to hop in an anticlockwise direction and skip alternate cells. The goal is to find out the last cell she would hop onto

Spiral Order traversal of Matrix(anti-clockwise),从左上角开始，一次往前走两步，返回最终能到的那个元素：类比 https://leetcode.com/problems/spiral-matrix/（改逆时针，走两步 ）

input: int N, int M, int[][] matrix

output: the integer of the last cell Lucy would hop onto



Example

Input:

3 3

29 8 37

15 41 3

1 10 14

Output:

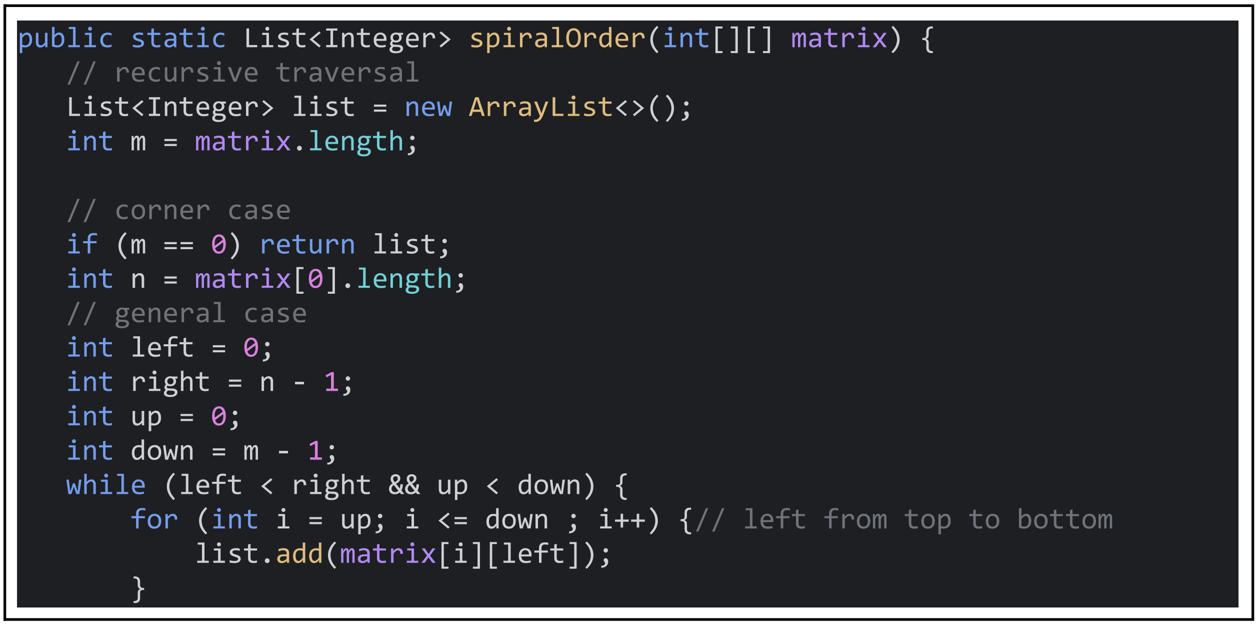
41

Explanation:

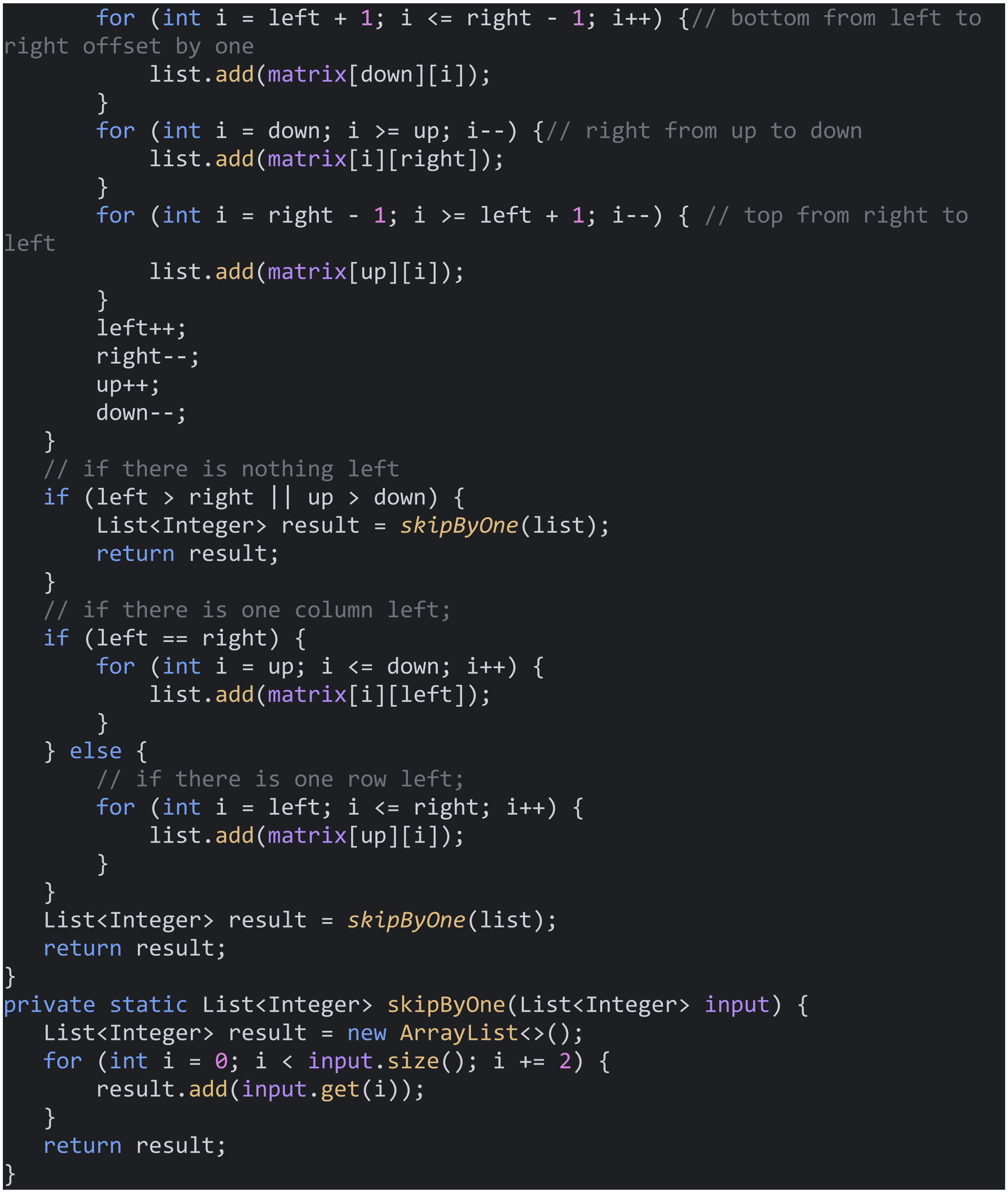
Lucy starts with 29, skips 15, hops onto 1, skip 10, hops onto 14, skips 3, hops onto 37, skips 8 and finally hops onto 41.

So, the output is 41.

** implementation



```
public static List<Integer> spiralOrder(int[][] matrix) {
   // recursive traversal
   List<Integer> list = new ArrayList<>();
   int m = matrix.length;

   // corner case
   if (m == 0) return list;
   int n = matrix[0].length;
   // general case
   int left = 0;
   int right = n - 1;
   int up = 0;
   int down = m - 1;
   while (left < right && up < down) {
       for (int i = up; i <= down ; i++) {// left from top to bottom
           list.add(matrix[i][left]);
       }
```



```
       for (int i = left + 1; i <= right - 1; i++) {// bottom from left to
right offset by one
           list.add(matrix[down][i]);
       }
       for (int i = down; i >= up; i--) {// right from up to down
           list.add(matrix[i][right]);
       }
       for (int i = right - 1; i >= left + 1; i--) { // top from right to
left
           list.add(matrix[up][i]);
       }
       left++;
       right--;
       up++;
       down--;
   }
   // if there is nothing left
   if (left > right || up > down) {
       List<Integer> result = skipByOne(list);
       return result;
   }
   // if there is one column left;
   if (left == right) {
       for (int i = up; i <= down; i++) {
           list.add(matrix[i][left]);
       }
   } else {
       // if there is one row left;
       for (int i = left; i <= right; i++) {
           list.add(matrix[up][i]);
       }
   }
   List<Integer> result = skipByOne(list);
   return result;
}
private static List<Integer> skipByOne(List<Integer> input) {
   List<Integer> result = new ArrayList<>();
   for (int i = 0; i < input.size(); i += 2) {
       result.add(input.get(i));
   }
   return result;
}
```

Version 2

Q1. Given an integer X, write an algorithm to find the number of integers which are less than or equal to X and whose digits add up to Y

input: int X, int Y

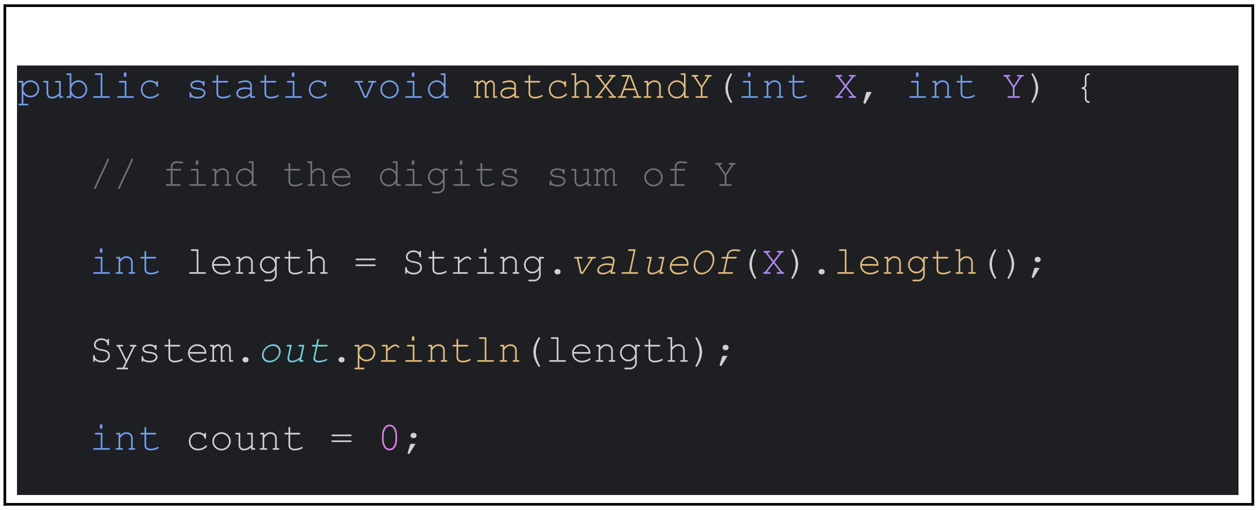
output: int number of integers

example: 20 5

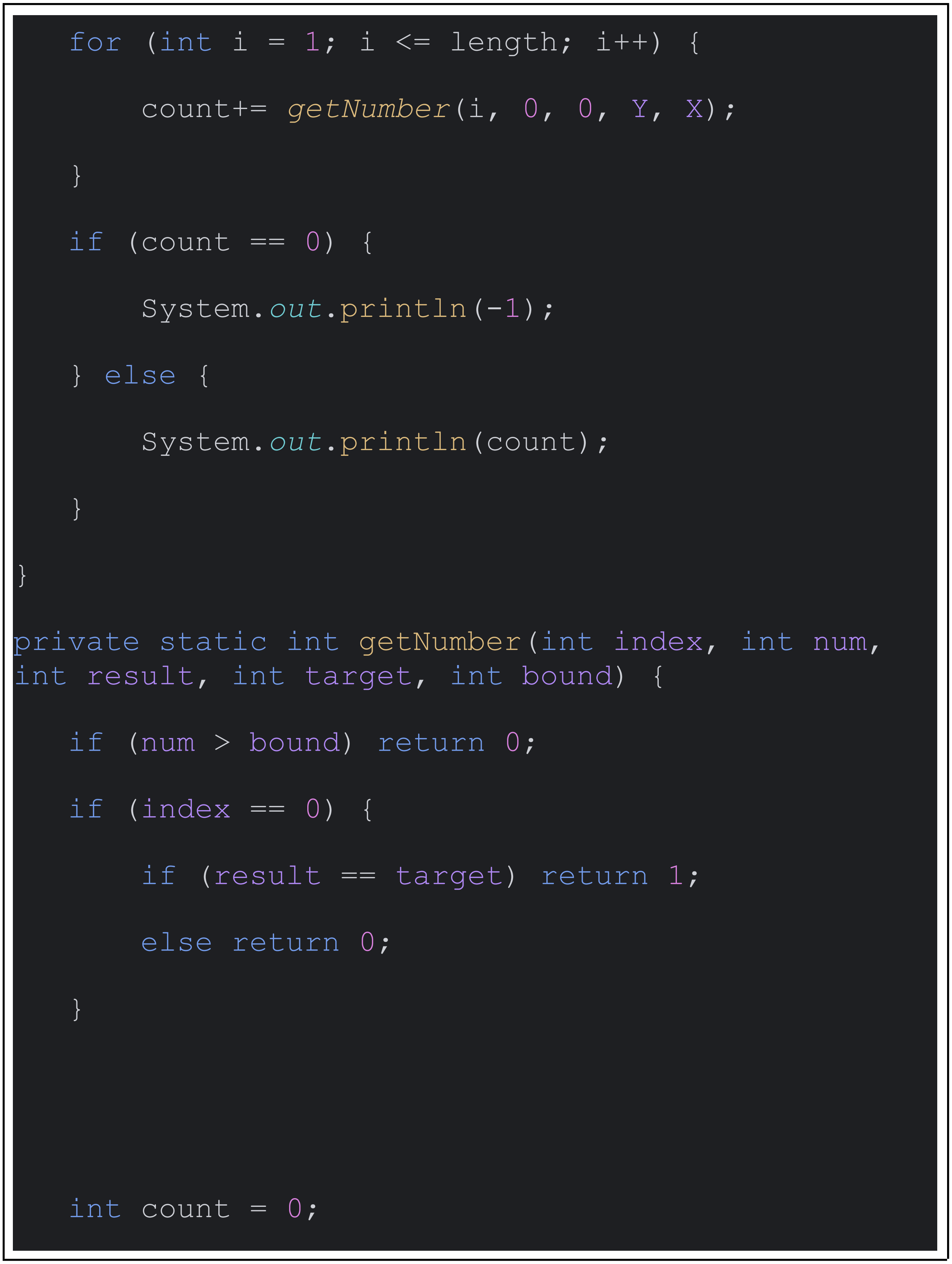
there are 2 two integers <= 20 and digits add up to 5

5 and 14, the answer is 2

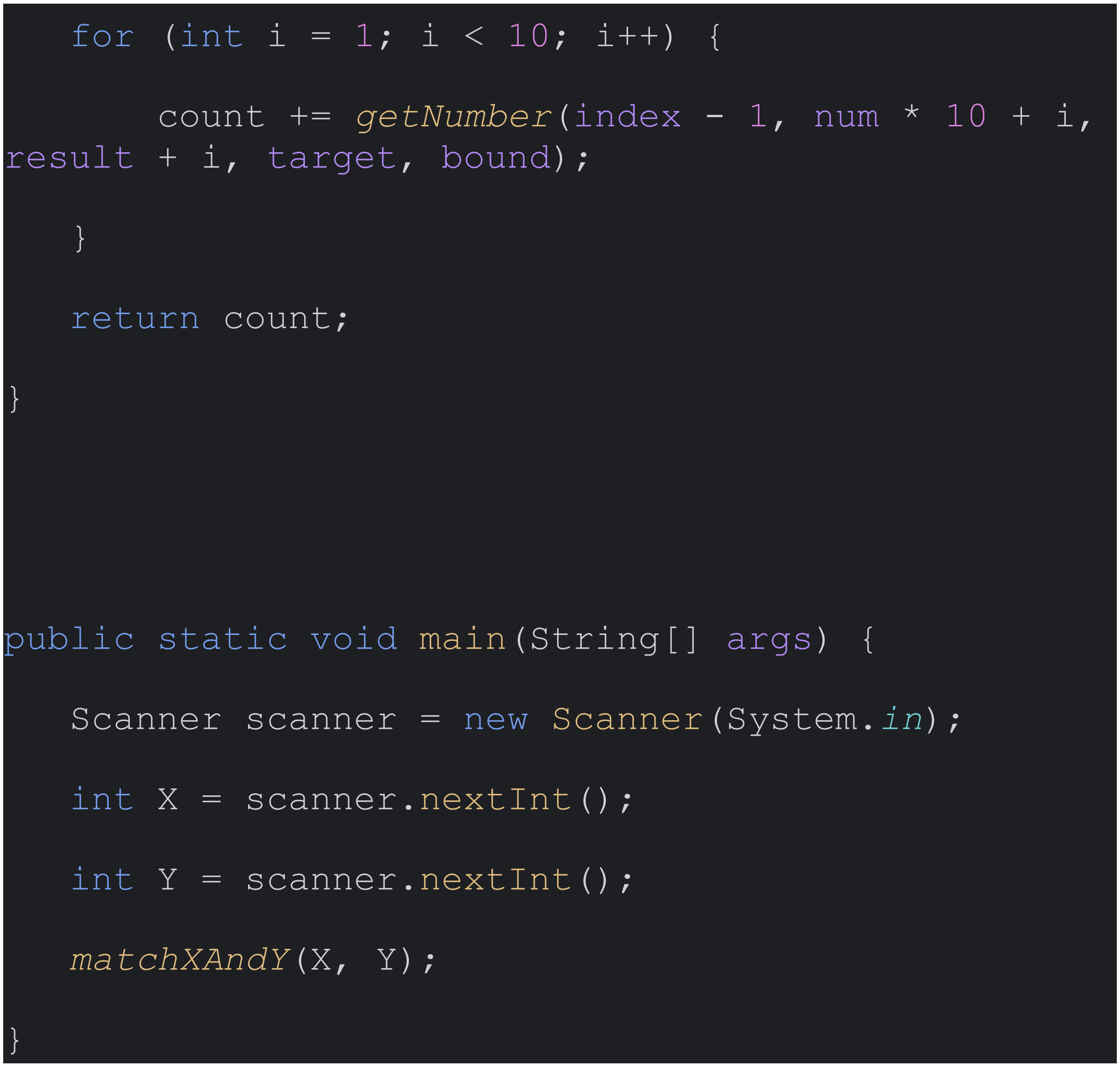
** implementation



```
public static void matchXAndY(int X, int Y) {
    // find the digits sum of Y
    int length = String.valueOf(X).length();
    System.out.println(length);
    int count = 0;
```



```
    for (int i = 1; i <= length; i++) {

        count+= getNumber(i, 0, 0, Y, X);

    }

    if (count == 0) {

        System.out.println(-1);

    } else {

        System.out.println(count);

    }

}

private static int getNumber(int index, int num, int result, int target, int bound) {

    if (num > bound) return 0;

    if (index == 0) {

        if (result == target) return 1;

        else return 0;

    }



    int count = 0;
```



```
    for (int i = 1; i < 10; i++) {
        count += getNumber(index - 1, num * 10 + i,
result + i, target, bound);
    }
    return count;
}



public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int X = scanner.nextInt();
    int Y = scanner.nextInt();
    matchXAndY(X, Y);
}
```

Q2. Write an algorithm to check if a string is sorted in alphabetical order and print 0 if it is. If it is not in alphabetical order, the print the index of the character where its is out of alphabetical order

input: String input

output: int 0 or index of char

example:

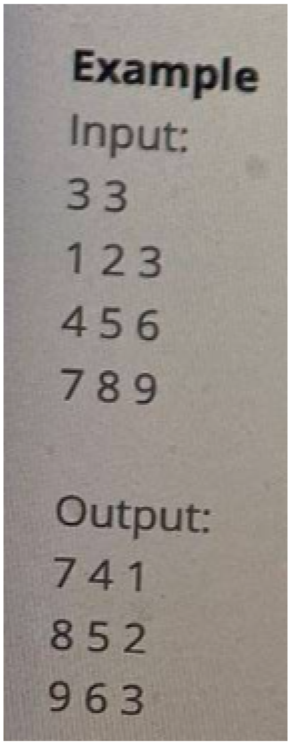
1. input abc   output: 0
2. input asd   output: 2 (这里应该是c++，index从1开始)

** implementation

```
public static int isInorder(String s) {

      // corner case

      if (s == null || s.length() == 0) return 0;

      if (s.length() == 1) return 0;

      // general case

      char[] array = s.toCharArray();

      char previous = array[0];
```

```
    for (int i = 1; i < array.length; i++) {

        if (array[i] < previous) return i;

        previous = array[i];

    }

    return 0;

}
```

Q3. You are given a matrix N*M size, write an algorithm to rotate the matrix by 90-degree



Example

Input:

```
3 3
1 2 3
4 5 6
7 8 9
```

Output:

```
7 4 1
8 5 2
9 6 3
```

```
public void rotate(int[][] matrix) {
        int n = matrix.length;
        // corner case
        if (n <= 1) return;
        // general case
        int round = n / 2;
        for (int level = 0; level < round; level++) {
            int left = level;
            int right = n - 2 - level;
            for (int i = left; i <= right; i++) {
                int temp = matrix[left][i];
                matrix[left][i] = matrix[n - 1- i][left];
                matrix[n - 1- i][left] = matrix[n - 1 - left][n - 1 - i];
                matrix[n - 1 - left][n - 1 - i] = matrix[i][n - 1 - left];
                matrix[i][n - 1 - left] = temp;
            }
        }
    }
```

Version 3

Q1. Fizz buzz

https://leetcode.com/problems/fizz-buzz/

** implementation

```
public List<String> fizzBuzz(int n) {
        List<String> list = new ArrayList<>();
        for (int i = 1; i <= n; i++) {
            if (i % 3 == 0 && i % 5 == 0) {
                list.add("FizzBuzz");
            } else if (i % 3 == 0) {
                list.add("Fizz");
            } else if (i % 5 == 0) {
                list.add("Buzz");
            } else {
                list.add(String.valueOf(i));
            }
        }
        return list;
    }
```

Q2. A pilot was asked to drop food packets in a terrain. He must fly over the entire terrain only once but cover a maximum number of drop points. The points are given as inputs in the form of integer coordinates in a 2-d field. The flight path can be horizontal or vertical, but not a mix of the two or diagonal

input: int xCorrdinateSize(representing the number of x coordinates), int[] drop point的x坐标

int yCorrdinateSize(representing the number of x coordinates, 和x的一定相等), int[] drop point的y坐标

output:int 最多cover的点

example：

input: 5

2 3 2 4 2

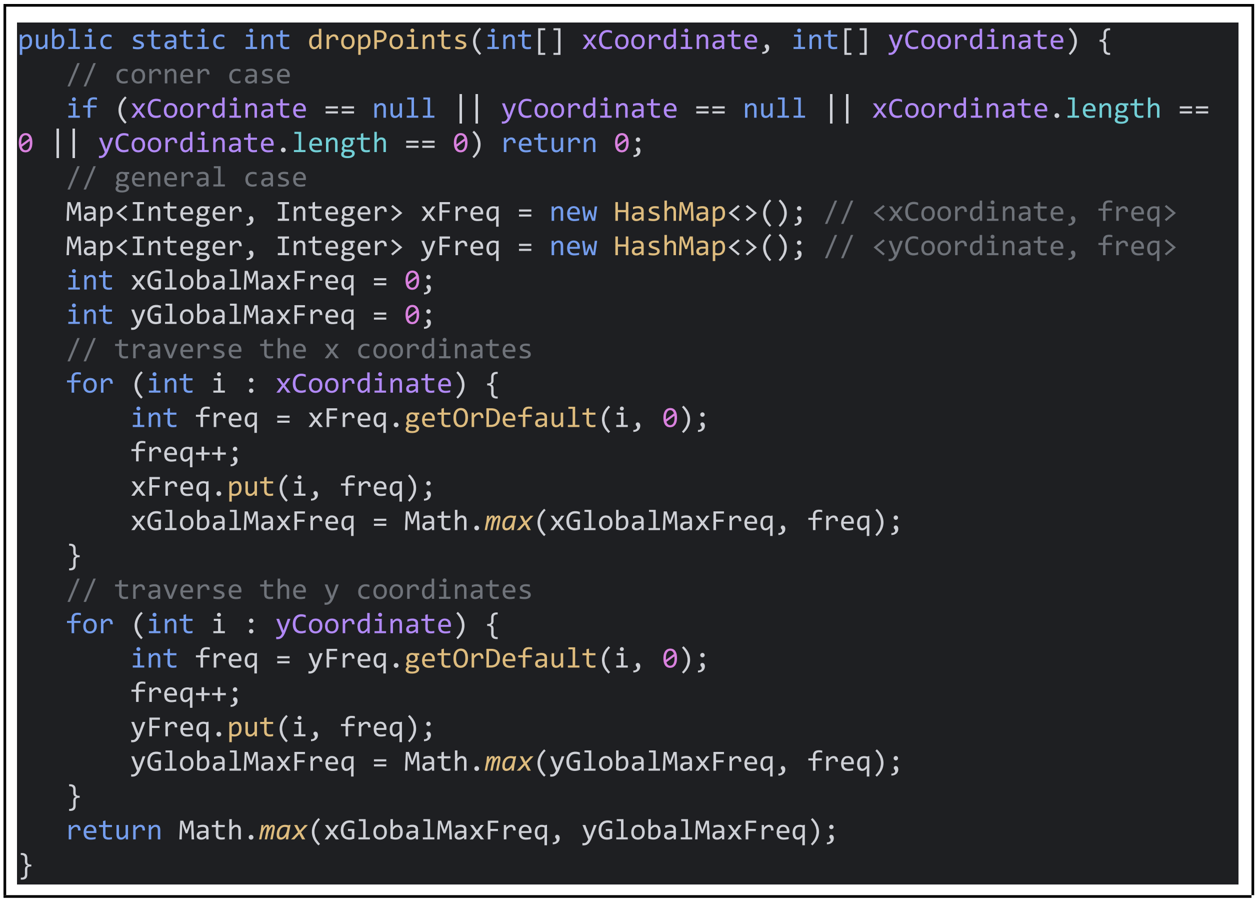
5

2 2 6 5 8

output: 3

5个drop points (2,2)(3,2)(2,6)(4,5)(2,8)

最大横着走完一行，(2,2)，(2,6)，(2,8)



```java
public static int dropPoints(int[] xCoordinate, int[] yCoordinate) {
   // corner case
   if (xCoordinate == null || yCoordinate == null || xCoordinate.length ==
0 || yCoordinate.length == 0) return 0;
   // general case
   Map<Integer, Integer> xFreq = new HashMap<>(); // <xCoordinate, freq>
   Map<Integer, Integer> yFreq = new HashMap<>(); // <yCoordinate, freq>
   int xGlobalMaxFreq = 0;
   int yGlobalMaxFreq = 0;
   // traverse the x coordinates
   for (int i : xCoordinate) {
       int freq = xFreq.getOrDefault(i, 0);
       freq++;
       xFreq.put(i, freq);
       xGlobalMaxFreq = Math.max(xGlobalMaxFreq, freq);
   }
   // traverse the y coordinates
   for (int i : yCoordinate) {
       int freq = yFreq.getOrDefault(i, 0);
       freq++;
       yFreq.put(i, freq);
       yGlobalMaxFreq = Math.max(yGlobalMaxFreq, freq);
   }
   return Math.max(xGlobalMaxFreq, yGlobalMaxFreq);
}
```

Q3. You are given a list of Integers(both positive and negative). Find the continuous sequence of integers with the largest sum

最大substring和

input: int arraySize, int[] array of Integer

output: largest sum

example:

input: 6

2, -8, 3, -2, 4, -10

output: 5

3 + (-2) + 4 = 5

** implementation

```
public int maxSubArray(int[] nums) {
        // corner case
        if (nums == null || nums.length == 0) return -1;
        // general case
        int[] dp = new int[nums.length];
        int globalMax = nums[0];
        dp[0] = nums[0];
        for (int i = 1; i < nums.length; i++) {
            // case 1 if not negative, add
            if (dp[i - 1] >= 0) {
                dp[i] = dp[i - 1] + nums[i];
            } else {
                dp[i] = nums[i];
            }
            globalMax = Math.max(globalMax, dp[i]);
        }
        return globalMax;
    }
```

Version 4.

Q1. write an algorithm to print a chessboard pattern(“B” for black squares, “W” for white sqaures), the top left is always white

类似8 Queens，填填填

input: int size of chessboard

output: List<List<character>>

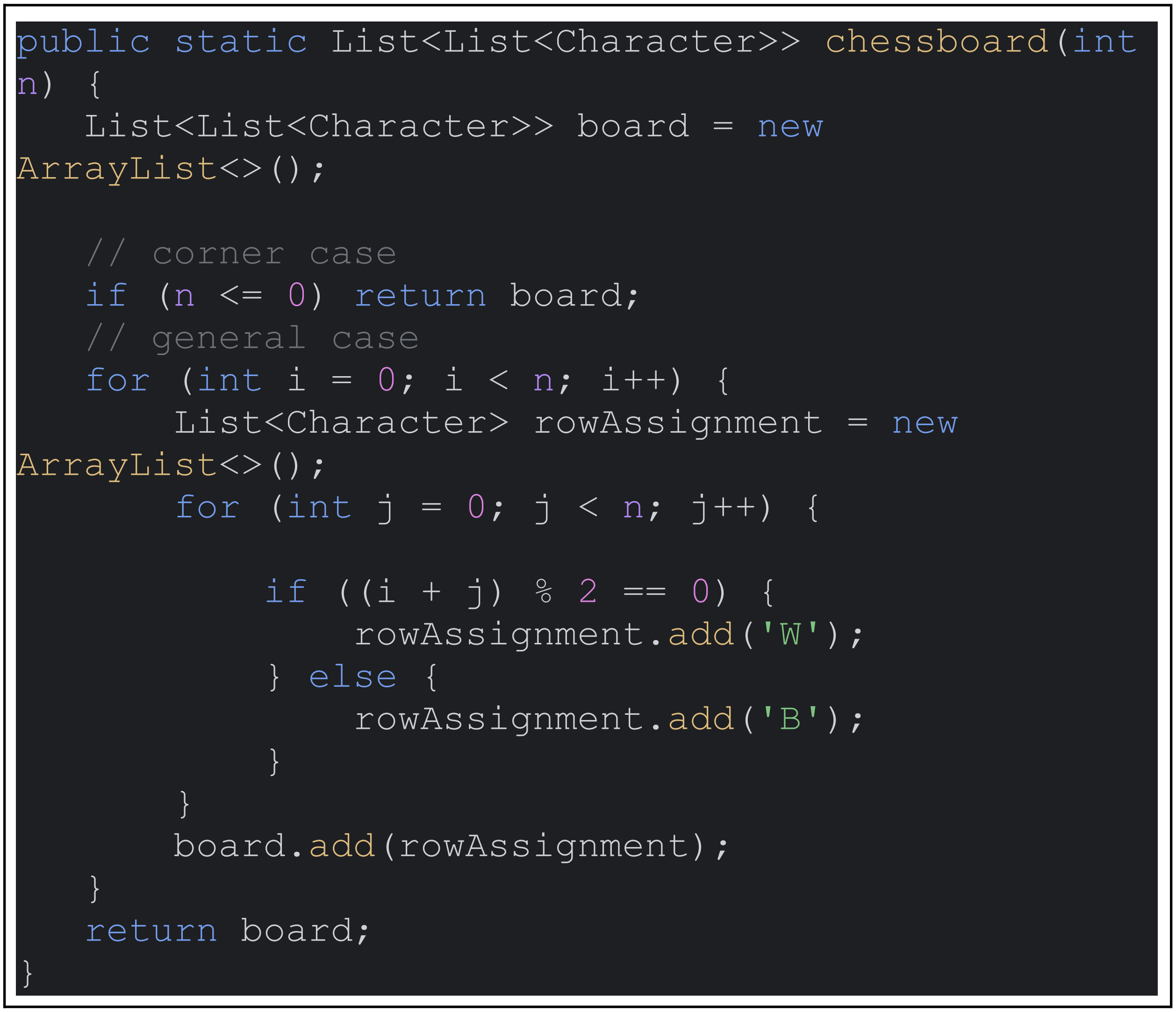
Example

Input:

5

Output:

```
W B W B W
B W B W B
W B W B W
B W B W B
W B W B W
```



```java
public static List<List<Character>> chessboard(int n) {
   List<List<Character>> board = new ArrayList<>();

   // corner case
   if (n <= 0) return board;
   // general case
   for (int i = 0; i < n; i++) {
       List<Character> rowAssignment = new ArrayList<>();
       for (int j = 0; j < n; j++) {

           if ((i + j) % 2 == 0) {
               rowAssignment.add('W');
           } else {
               rowAssignment.add('B');
           }
       }
       board.add(rowAssignment);
   }
   return board;
}
```

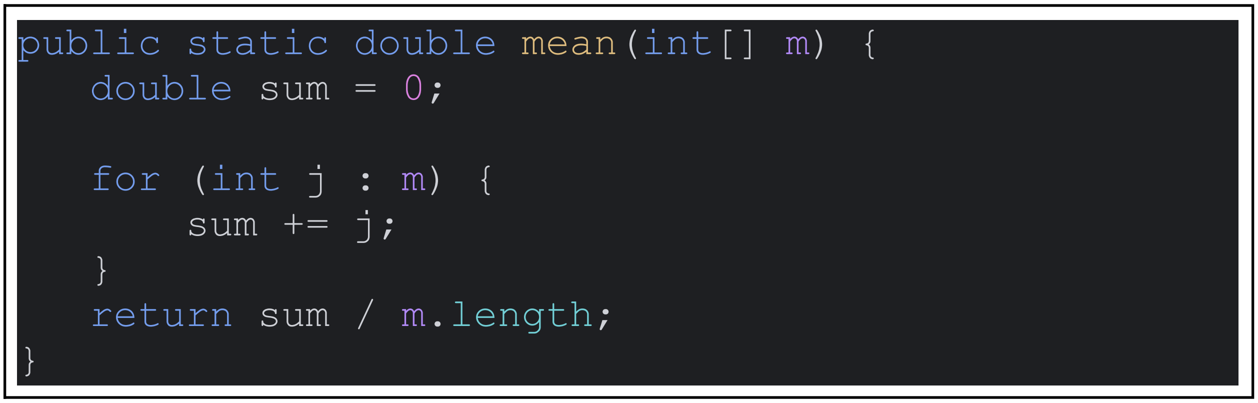
Q2. The arithmetic mean of N numbers is the number of the sum of all the numbers divided by N. The mode of N numbers is the most frequently occurring number. Write an algorithm to find the mean and mode of a set of given number

input: int numberOfInteger, int[] numbers

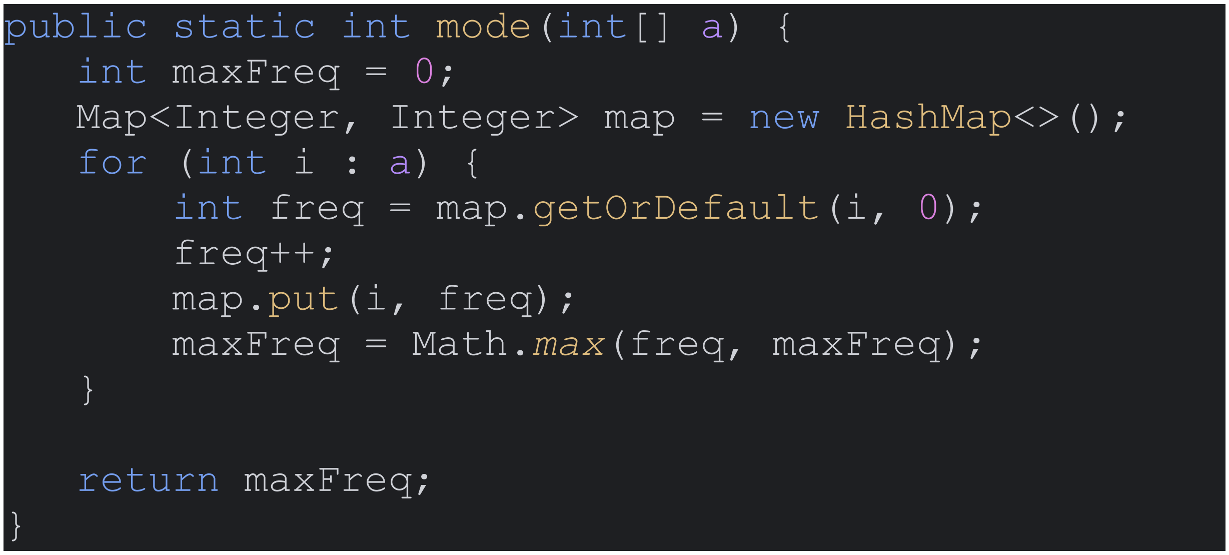
output: mean and mode

example

```
    input: 5

        [1 2 7 3 2]

    output: mean: 3

            mode: 2
```



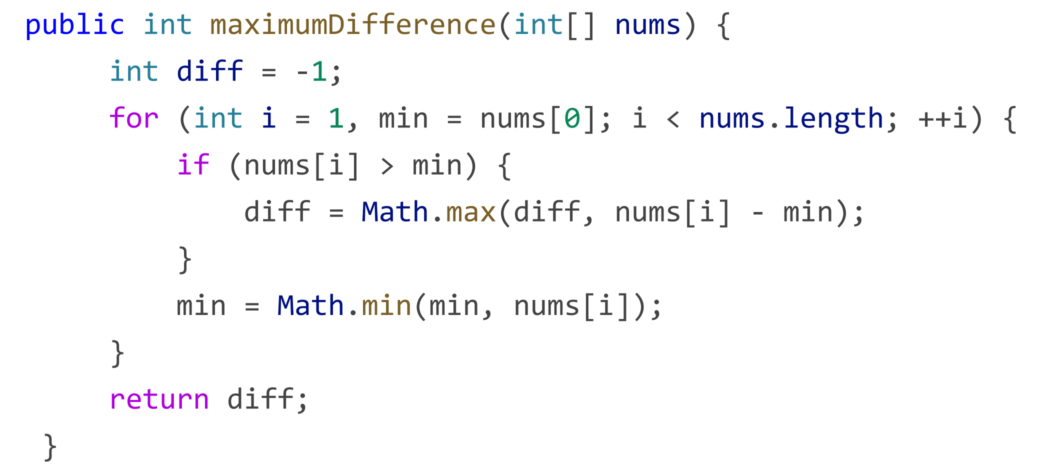
```
public static double mean(int[] m) {
   double sum = 0;

   for (int j : m) {
       sum += j;
   }
   return sum / m.length;
}
```



```java
public static int mode(int[] a) {
    int maxFreq = 0;
    Map<Integer, Integer> map = new HashMap<>();
    for (int i : a) {
        int freq = map.getOrDefault(i, 0);
        freq++;
        map.put(i, freq);
        maxFreq = Math.max(freq, maxFreq);
    }

    return maxFreq;
}
```

Q3．递增index的最大差值

https://leetcode.com/problems/maximum-difference-between-increasing-elements/



```java
public int maximumDifference(int[] nums) {
    int diff = -1;
    for (int i = 1, min = nums[0]; i < nums.length; ++i) {
        if (nums[i] > min) {
            diff = Math.max(diff, nums[i] - min);
        }
        min = Math.min(min, nums[i]);
    }
    return diff;
 }
```

Version 5

Q1. Ray, Shiv and Ansh are conducting a survey for a group of people. The survey is only meant for twins but there are certain people who are not twins and wanting to take part in the survey. Write an algorithm to help them identify the person from the given input who is not a twin

input: int size, int[] givenIntegers

output: int 最小的单独数

7

1 1 2 3 3 4 4

Output:

2

Explanation:

In the given array of element, only nontwin element is '2So, the output is 2

Example 2:

Input:

4

1122

Output:

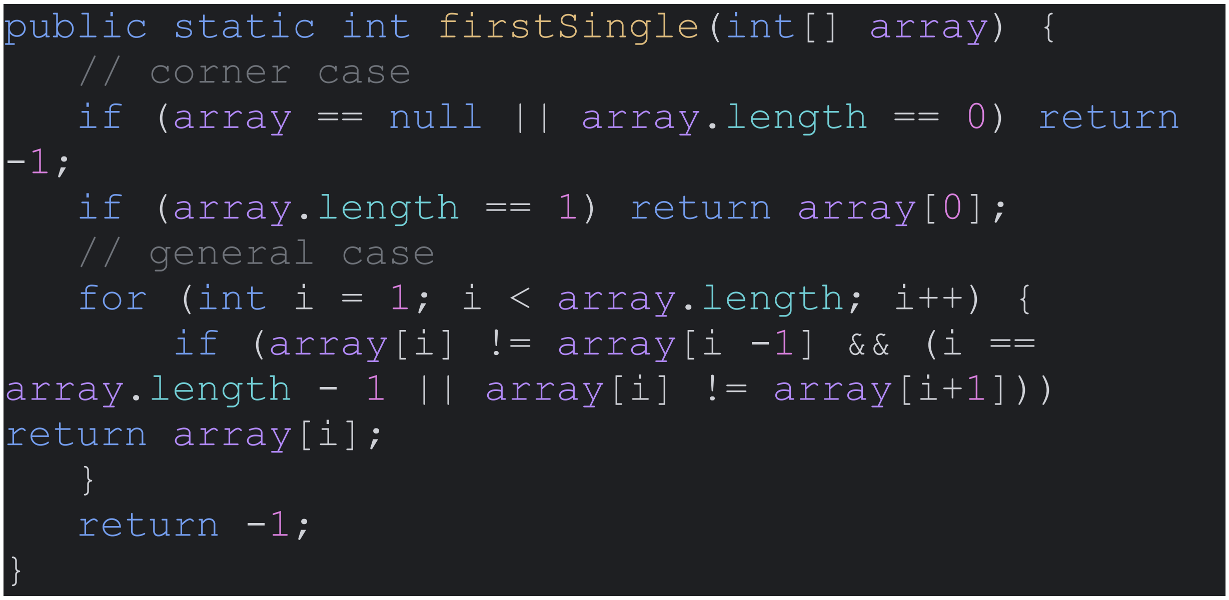
-1

Explanation:

Given array of element contain all the

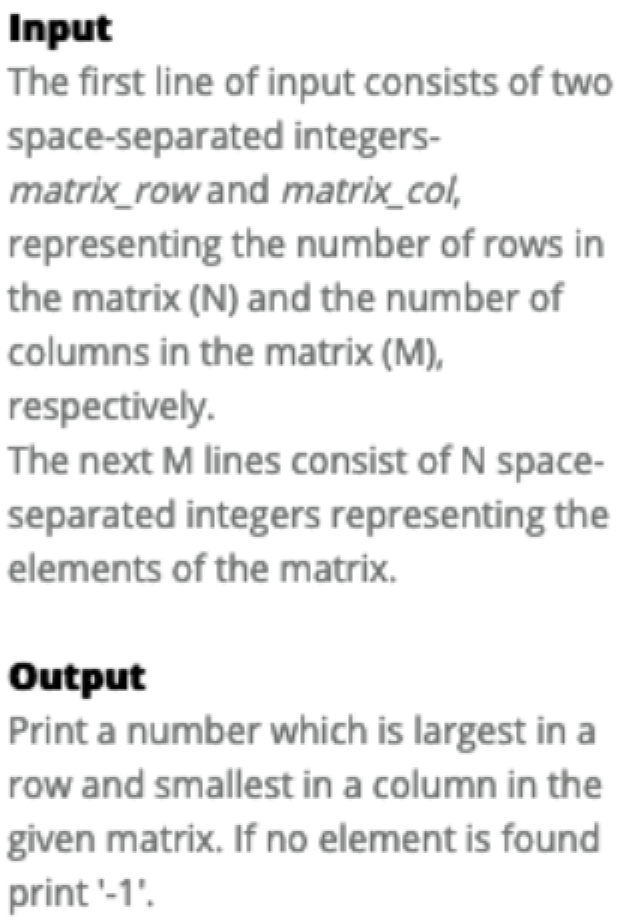
twin elements.

So, the output is -1.



```
public static int firstSingle(int[] array) {
    // corner case
    if (array == null || array.length == 0) return
-1;
    if (array.length == 1) return array[0];
    // general case
    for (int i = 1; i < array.length; i++) {
        if (array[i] != array[i -1] && (i ==
array.length - 1 || array[i] != array[i+1]))
return array[i];
    }
    return -1;
}
```

Q2.Write an algorithm which finds out the elements which are largest in a row and smallest in a column in a matrix



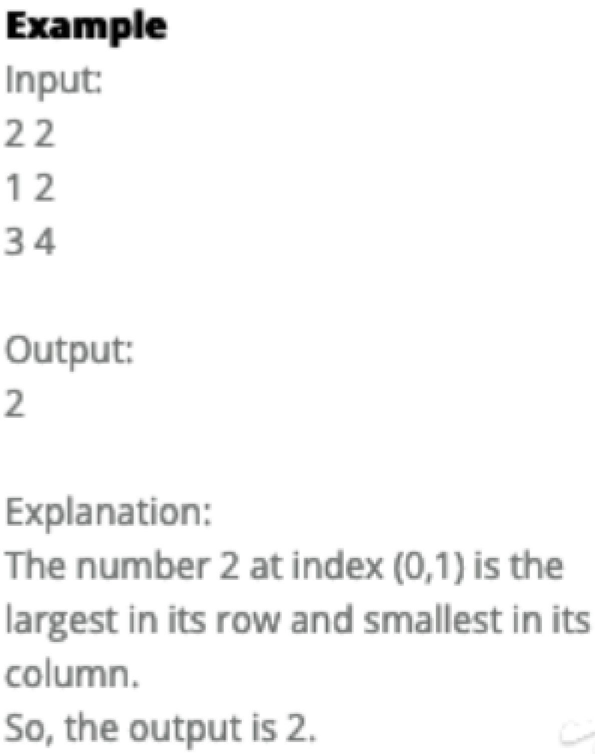
Input

The first line of input consists of two space-separated integers- *matrix_row* and *matrix_col*, representing the number of rows in the matrix (N) and the number of columns in the matrix (M), respectively.

The next M lines consist of N space-separated integers representing the elements of the matrix.

Output

Print a number which is largest in a row and smallest in a column in the given matrix. If no element is found print '-1'.



Example

Input:

2 2

1 2

3 4

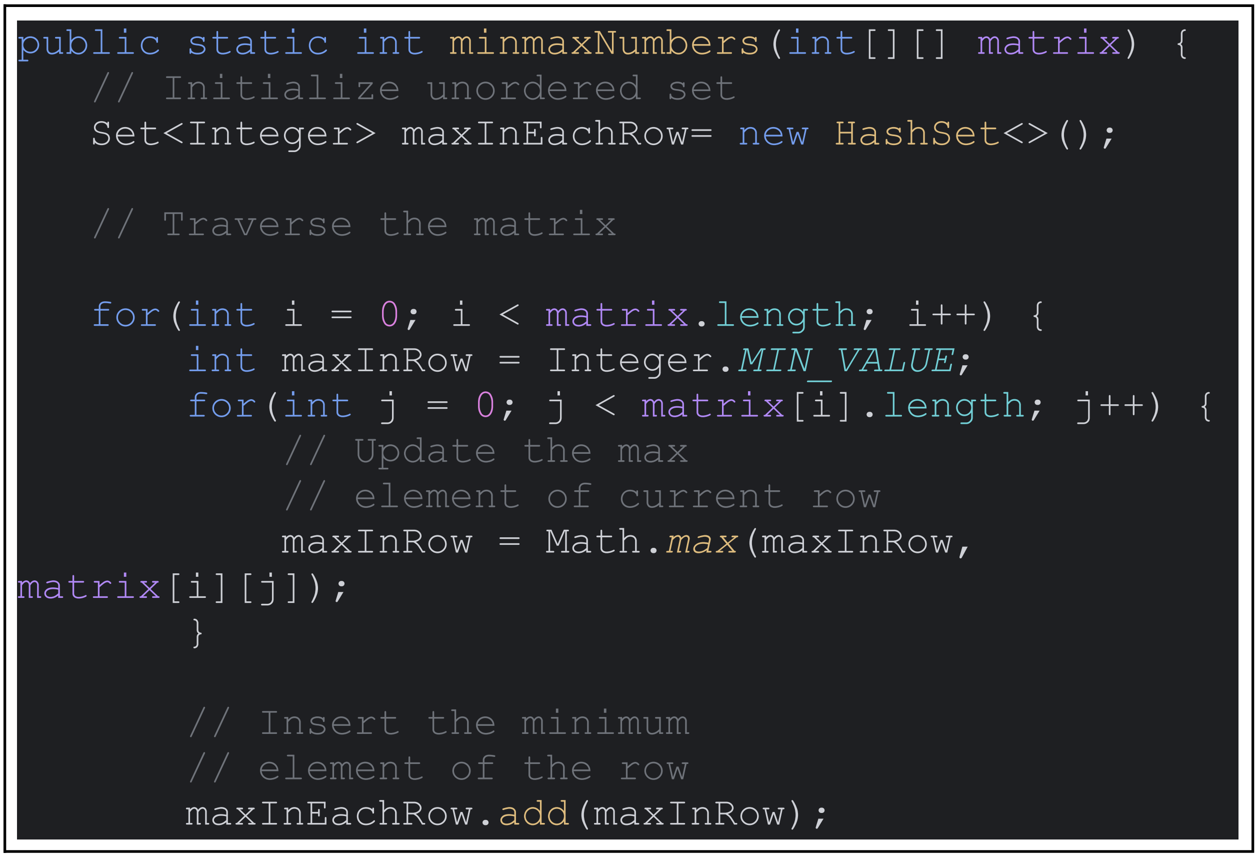
Output:

2

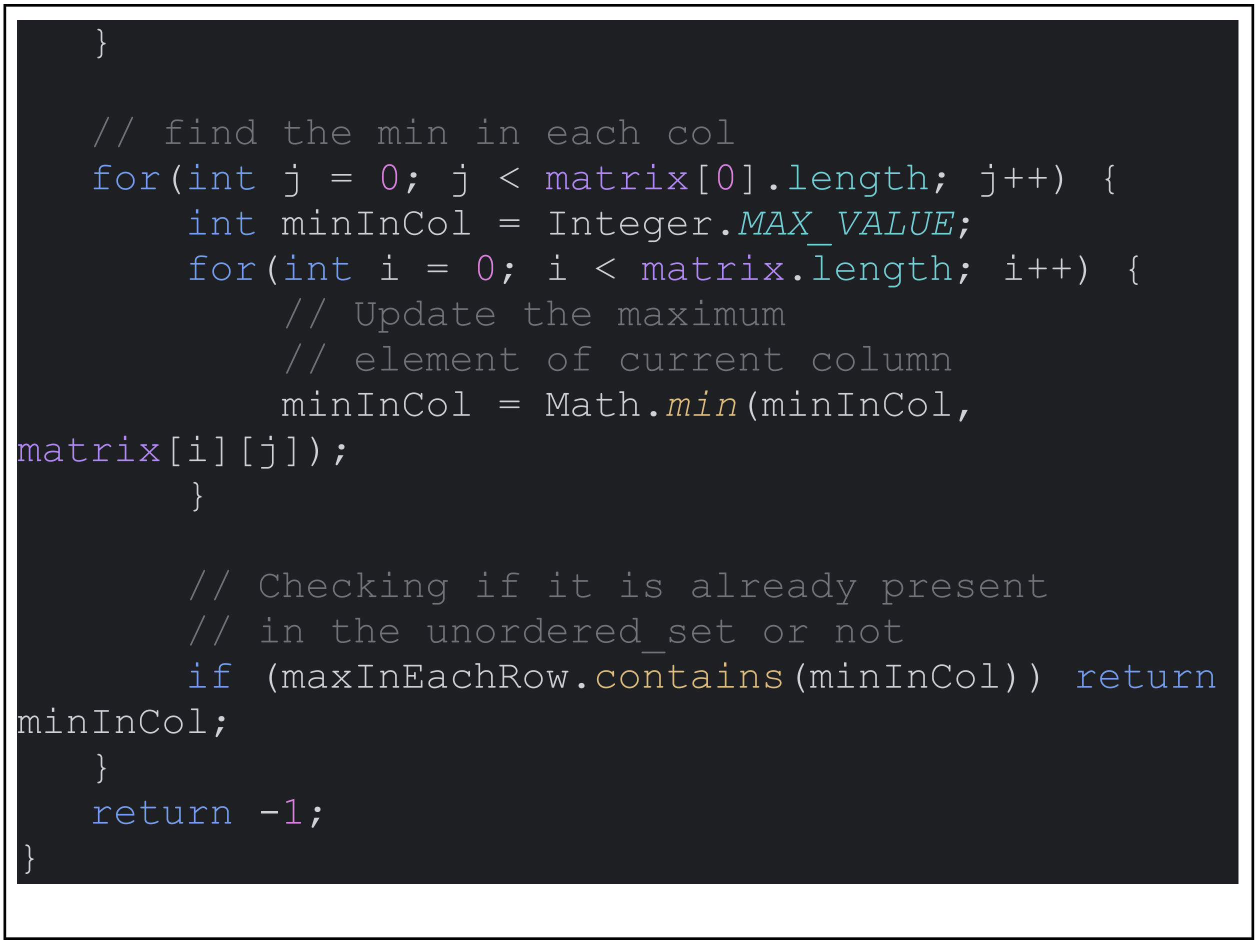
Explanation:

The number 2 at index (0,1) is the largest in its row and smallest in its column.

So, the output is 2.

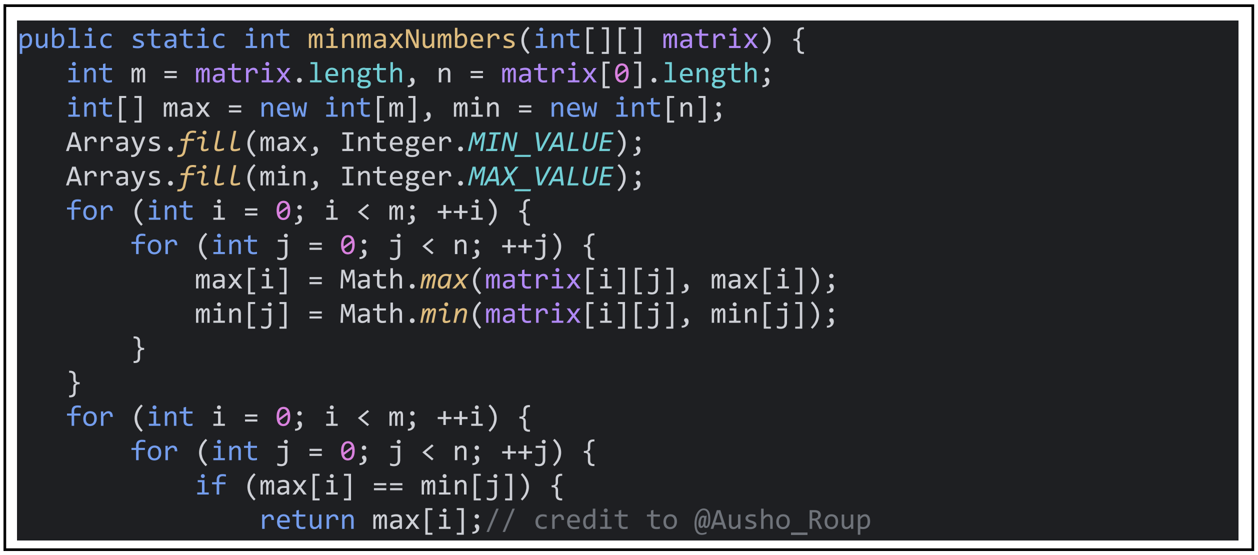


```
public static int minmaxNumbers(int[][] matrix) {
    // Initialize unordered set
    Set<Integer> maxInEachRow= new HashSet<>();

    // Traverse the matrix

    for(int i = 0; i < matrix.length; i++) {
         int maxInRow = Integer.MIN_VALUE;
         for(int j = 0; j < matrix[i].length; j++) {
              // Update the max
              // element of current row
              maxInRow = Math.max(maxInRow,
matrix[i][j]);
         }

         // Insert the minimum
         // element of the row
         maxInEachRow.add(maxInRow);
```



```
    }

    // find the min in each col
    for(int j = 0; j < matrix[0].length; j++) {
        int minInCol = Integer.MAX_VALUE;
        for(int i = 0; i < matrix.length; i++) {
            // Update the maximum
            // element of current column
            minInCol = Math.min(minInCol,
matrix[i][j]);
        }

        // Checking if it is already present
        // in the unordered_set or not
        if (maxInEachRow.contains(minInCol)) return
minInCol;
    }
    return -1;
}
```

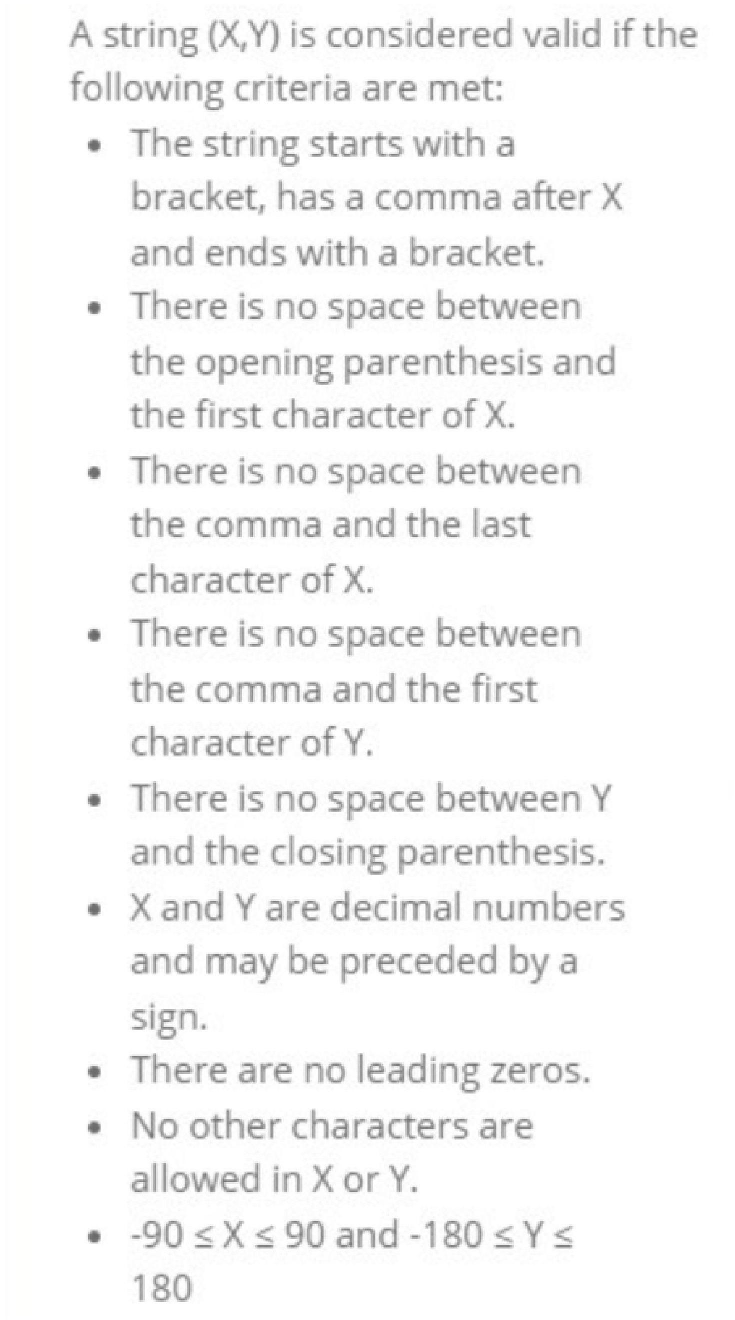
method 2



```
public static int minmaxNumbers(int[][] matrix) {
   int m = matrix.length, n = matrix[0].length;
   int[] max = new int[m], min = new int[n];
   Arrays.fill(max, Integer.MIN_VALUE);
   Arrays.fill(min, Integer.MAX_VALUE);
   for (int i = 0; i < m; ++i) {
       for (int j = 0; j < n; ++j) {
           max[i] = Math.max(matrix[i][j], max[i]);
           min[j] = Math.min(matrix[i][j], min[j]);
       }
   }
   for (int i = 0; i < m; ++i) {
       for (int j = 0; j < n; ++j) {
           if (max[i] == min[j]) {
               return max[i];// credit to @Ausho_Roup
```

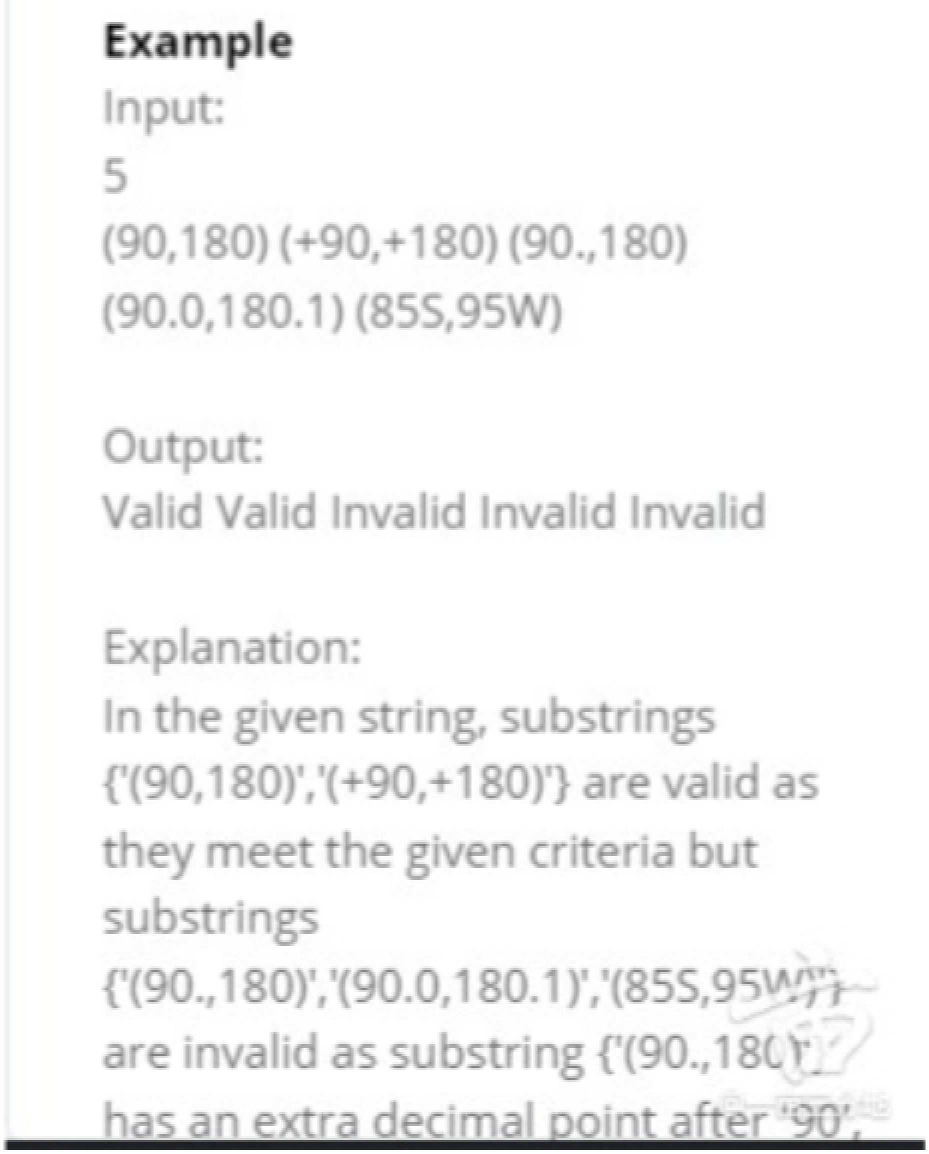
```
            }
        }
    }
    return -1;
}
```

Q3. You are given a list of Strings that may represent valid latitude/longitude pais. Your task is to check if the given pairs are valid or not



A string (X,Y) is considered valid if the following criteria are met:

- The string starts with a bracket, has a comma after X and ends with a bracket.
- There is no space between the opening parenthesis and the first character of X.
- There is no space between the comma and the last character of X.
- There is no space between the comma and the first character of Y.
- There is no space between Y and the closing parenthesis.
- X and Y are decimal numbers and may be preceded by a sign.
- There are no leading zeros.
- No other characters are allowed in X or Y.
- -90 ≤ X ≤ 90 and -180 ≤ Y ≤ 180



Example

Input:

5

(90,180) (+90,+180) (90.,180)

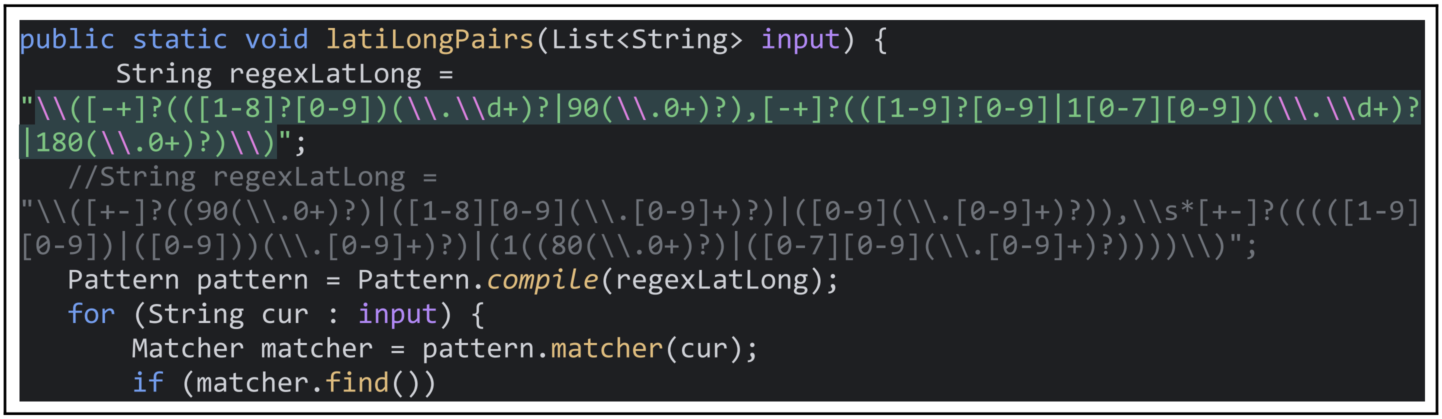
(90.0,180.1) (85S,95W)

Output:

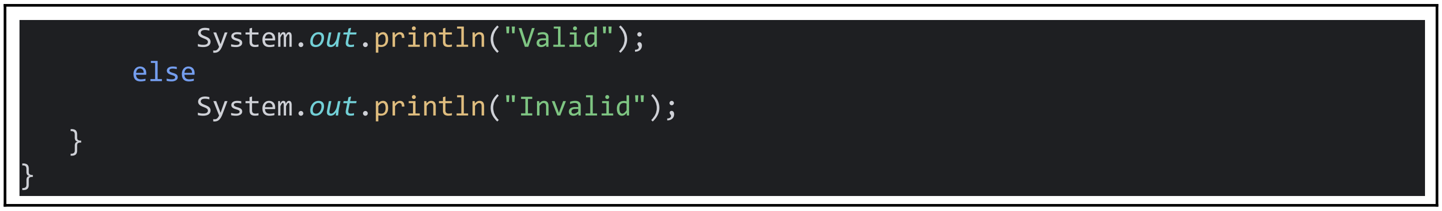
Valid Valid Invalid Invalid Invalid

Explanation:

In the given string, substrings {'(90,180)','(+90,+180)'} are valid as they meet the given criteria but substrings {'(90.,180)','(90.0,180.1)','(85S,95W)'} are invalid as substring {'(90.,180)'} has an extra decimal point after '90',

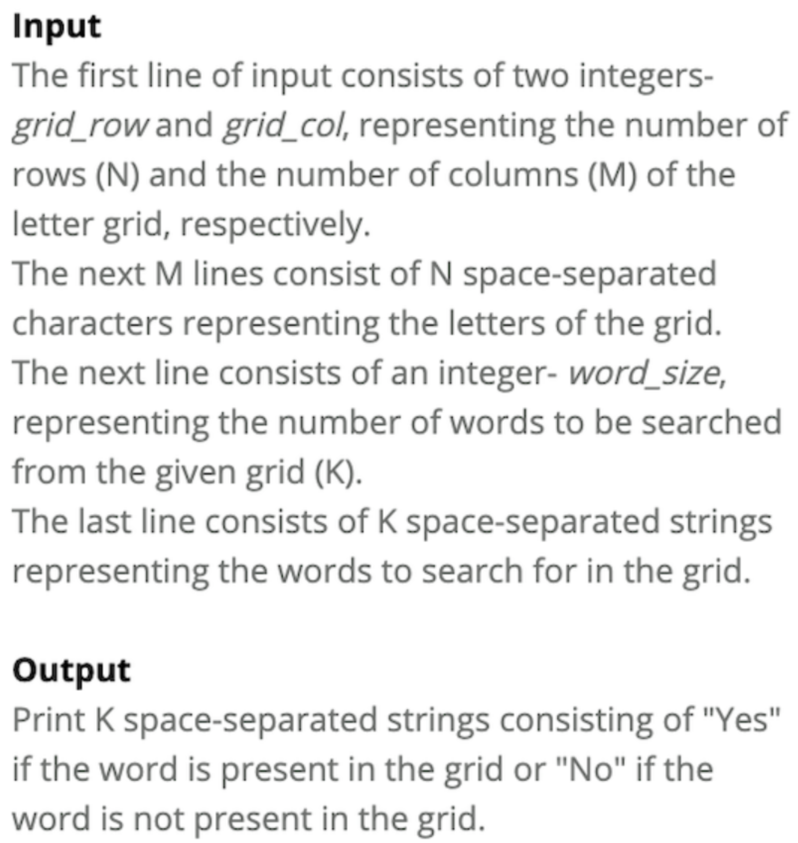


```
public static void latiLongPairs(List<String> input) {
      String regexLatLong =
"\\([-+]?(([1-8]?[0-9])(\\.\\d+)?|90(\\.0+)?),[-+]?(([1-9]?[0-9]|1[0-7][0-9])(\\.\\d+)?
|180(\\.0+)?)\\)";
   //String regexLatLong =
"\\([+-]?((90(\\.0+)?)|([1-8][0-9](\\.[0-9]+)?)|([0-9](\\.[0-9]+)?)),\\s*[+-]?((([1-9]
[0-9])|([0-9]))(\\.[0-9]+)?)|(1((80(\\.0+)?)|([0-7][0-9](\\.[0-9]+)?))))\\)";
   Pattern pattern = Pattern.compile(regexLatLong);
   for (String cur : input) {
       Matcher matcher = pattern.matcher(cur);
       if (matcher.find())
```



```
           System.out.println("Valid");
       else
           System.out.println("Invalid");
   }
}
```

Q4. You are given a grid of letters, followed by some words. The words can occur anywhere in the grid on a row or a column, forward or backwards. However, there are no diagonal words. Write an algorithm to find if the given word occurs in the grid on a row or a column, forward or backwards

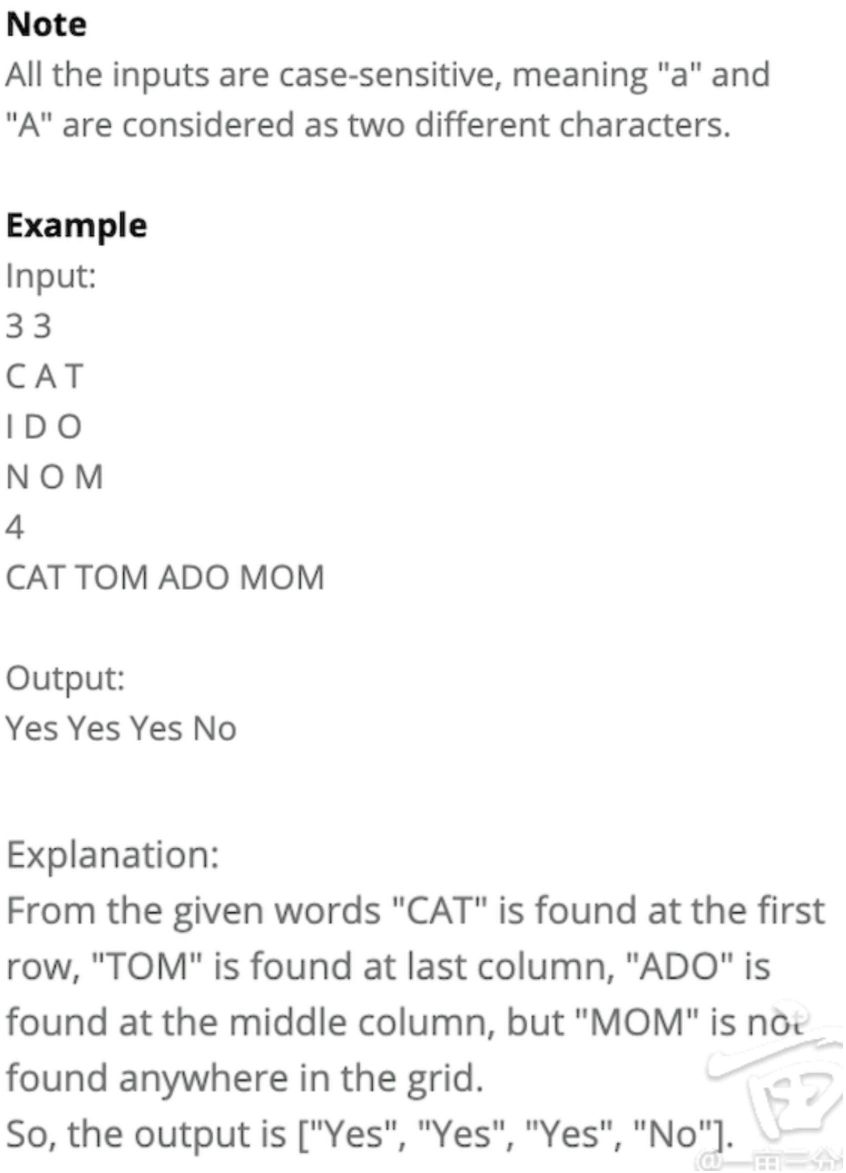


Input

The first line of input consists of two integers- *grid_row* and *grid_col*, representing the number of rows (N) and the number of columns (M) of the letter grid, respectively.
The next M lines consist of N space-separated characters representing the letters of the grid.
The next line consists of an integer- *word_size*, representing the number of words to be searched from the given grid (K).
The last line consists of K space-separated strings representing the words to search for in the grid.

Output

Print K space-separated strings consisting of "Yes" if the word is present in the grid or "No" if the word is not present in the grid.



Note

All the inputs are case-sensitive, meaning "a" and "A" are considered as two different characters.

Example

Input:
3 3
C A T
I D O
N O M
4
CAT TOM ADO MOM

Output:
Yes Yes Yes No

Explanation:
From the given words "CAT" is found at the first row, "TOM" is found at last column, "ADO" is found at the middle column, but "MOM" is not found anywhere in the grid.
So, the output is ["Yes", "Yes", "Yes", "No"].

```java
private static final int[][] DIRS = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
    public boolean exist(char[][] board, String word) {
        // corner case
        if (word == null || word.length() == 0) return false;
        boolean[][] visited = new boolean[board.length][board[0].length];
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                if (!visited[i][j] && board[i][j] == word.charAt(0)) {
                    if (dfs(board, word, visited, i, j, 0)) return true;
                }
```

```
      }
    }
    return false;
  }
  private boolean dfs(char[][] board, String word, boolean[][] visited,
int row, int col, int index) {
    // base case
    if (index == word.length()) return true;
    // check within bound
    if (row < 0 || row >= board.length || col < 0 || col >=
board[0].length) return false;
    // check visited;
    if (visited[row][col]) return false;
    // check match
    if (board[row][col] != word.charAt(index)) return false;
    // mark visited
    visited[row][col] = true;
    // do dfs
    for (int[] direction : DIRS) {
      int newRow = row + direction[0];
      int newCol = col + direction[1];
      if(dfs(board, word, visited, newRow, newCol, index + 1)) return
true;
    }
    // backtracking
    visited[row][col] = false;
    return false;
  }
```

method 2 Trietree

```
private static final int[][] DIRS = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
 static class TrieNode {
  TrieNode[] children = new TrieNode[26];
  boolean isWord;
 }
 public List<String> findWords(char[][] board, String[] words) {
   List<String> list = new ArrayList<>();
   // corner case
   if (board == null || board.length == 0 || board[0].length == 0)
return list;
   if (words == null || words.length == 0) return list;
   // general case
   Set<String> res = new HashSet<>();
   // preprocess to convert input string[] into TrieTree
   TrieNode root = buildTrietree(words);
   int rows = board.length;
   int cols = board[0].length;
   StringBuilder sb = new StringBuilder(); // to store prefix
   boolean[][] visited = new boolean[rows][cols];
   for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
     DFS(board, i, j, root, sb, visited, res);
    }
   }
   return new ArrayList<>(res);
 }
 // helper function to apply DFS
 private void DFS(char[][] board, int row, int col, TrieNode root,
StringBuilder sb, boolean[][] visited, Set<String> res) {
  // base case
  if (root.isWord) res.add(sb.toString());
  // check validity
  if (row < 0 || row >= board.length || col < 0 || col >= board[0].length
|| visited[row][col]) return;
  // check children in TrieTree
  char ch = board[row][col];
  if (root.children[ch - 'a'] == null) return;
  // if contains, run dfs
  root = root.children[ch - 'a'];
  visited[row][col] = true;
  sb.append(ch);
  for (int[] direction : DIRS) {
   int newRow = row + direction[0];
   int newCol = col + direction[1];
   DFS(board, newRow, newCol, root, sb, visited, res);
  }
  // backtracking
  sb.deleteCharAt(sb.length() - 1);
  visited[row][col] = false;
 }
 // helper function to build Trietree
 private TrieNode buildTrietree(String[] words) {
  TrieNode root = new TrieNode();
  for (String word : words) {
   TrieNode cur = root;
   // main logic traverse each word
   for (int i = 0; i < word.length(); i++) {
    TrieNode child = cur.children[word.charAt(i) - 'a'];
    if (child == null) { // if have not been create, create it
     child = new TrieNode();
     cur.children[word.charAt(i) - 'a'] = child;
    }
    cur = child; // update the cur pointer
   }
   cur.isWord = true; // once finish each word, mark it as word
```

```
    }
    return root;
  }
```