```java
/*
*  This file is the set up and solution to the lab final of Fall 2023.
*
*  by R. Heise
*  Dec 11, 2023 at 9:25:28 p.m.
*
*/
import java.util.LinkedList;
//only use: constructor, addFirst, addLast, removeFirst, removeLast, size,
//and isEmpty

public class Final {
    //=========================================================================
    //=========================================================================
    //=========================================================================
    //=========================================================================
    // Put your code here for nested class SearchSpot


    /**
     * Nested class called SearchSpot
     *
     * Default constructor:
     * public SearchSpot(int row, int column)
     * toString:
     * public String toString()
     */
    public static class SearchSpot {
    //Data field
    public int row;
    public int column;
    //Data field

    /**
     * Default constructor for SearchSpot
     * @param row
     * @param column
     */
    public SearchSpot(int row, int column) {
        this.row = row;
        this.column = column;
    }//default constructor

    @Override
    public String toString() {
        return "(" + row + ", " + column + ")";
    }
}//SearchSpot nested class




    //=========================================================================
    //=========================================================================ca
    // Put your code here for findTreasure

    /**
     * This method works with the nested class SearchSpot,
     * it goes through a 2D array/map with the goal of finding a row and column
     * value equivalent to value in row/col point on the array/map, for example
     * if the row is 5 and the column is 2, and the value at that point is 52,
     * then the treasure has been found. Returns the path taken in the form
     * of the data structure of a linked list
     * @param array The 2D array/map passed into the method that searches for
     * the treasure.
     * @return the path taken in order to find the treasure, as well as if it
     * has found the treasure or not. If one of the base cases are satisfied
     * it returns null as the path instead.
     */
    public static LinkedList<SearchSpot> findTreasure(int[][] array) {
     //if array size is null it returns null
     //base case
    if (array == null || array.length == 0) {
        System.out.println(x: "Treasure not found");
        return null;
    }//if

    LinkedList<SearchSpot> path = new LinkedList<>();
    //the limit variable limits the method to check over the array to one greater
    //than the size of the array.
    int limit = array.length * array.length + 1;
    //setting variables
    int row = 0;
    int column = 0;
    //number of checks
    int check = 0;


   //while loop to stop the method iterating over the array multiple times
   //rather it stops checking at the one plus the length of the array.
   //Being the limit variable
    while (check < limit) {
        if (row < 0 || row >= array.length || column < 0 || column >= array[0].length) {
            System.out.println(x: "Treasure not found");
            return null;
        }//if
        //getting each value for row and column
        int value = array[row][column];
        int rowClue = value / 10 - 1;
        int columnClue = value % 10 - 1;

        path.addLast(new SearchSpot(row + 1, column + 1));
        if (row == rowClue && column == columnClue) {
            System.out.println(x: "Found treasure");
            return path;
        }//if
        //sets row and column to where it currently is in the array.
        row = rowClue;
        column = columnClue;
        check++;
    }//while loop
    //If the code is checked for one more than the size of the it
    //array prints this
    System.out.println(x: "Treasure not found");
    return null;
}//FindTreasure method




    //=========================================================================
    //=========================================================================
    //=========================================================================
    //=========================================================================




    //=========================================================================
    //=========================================================================
    //Any code placed below here will not be evaluated towards your grade
    //on this lab exam
```

```
137     public static void main(String[] args) {
138         System.out.println(x: "Testing 1 2 3...");
139         testTreasure();
140         System.out.println(x: "\n========================================");
141         testSumMinMax();
142     }
143 
144     /**
145      * Tests a bunch of different square arrays for treasure (invariant
146      * points).
147      */
148     public static void testTreasure(){
149         LinkedList<SearchSpot> path;
150         int[][] map1 = {{34, 21, 32, 41, 25},
151                         {14, 42, 43, 14, 31},
152                         {54, 45, 52, 42, 23},
153                         {33, 15, 51, 31, 35},
154                         {21, 52, 33, 13, 23}};
155 
156         int[][] map2 = {{62, 13, 53, 14, 51, 66},
157                         {12, 25, 43, 31, 56, 42},
158                         {55, 42, 36, 22, 26, 54},
159                         {51, 34, 45, 61, 13, 22},
160                         {51, 44, 36, 54, 25, 66},
161                         {54, 35, 63, 15, 21, 45}};
162 
163         int[][] map3 = {{11, 31, 22},
164                         {32, 12, 21},
165                         {13, 32, 23}};
166 
167         int[][] map4 = {{55, 12, 25, 37, 78, 19, 81, 43, 83},
168                         {14, 23, 48, 59, 93, 52, 59, 96, 54},
169                         {72, 65, 19, 97, 35, 49, 41, 76, 13},
170                         {48, 65, 57, 44, 85, 24, 17, 15, 19},
171                         {79, 61, 65, 58, 61, 32, 72, 41, 18},
172                         {82, 39, 31, 53, 74, 58, 67, 32, 69},
173                         {15, 29, 28, 74, 99, 75, 17, 78, 56},
174                         {29, 33, 85, 11, 22, 67, 87, 94, 43},
175                         {91, 81, 95, 69, 76, 35, 16, 98, 21}};
176 
177         int[][] map5 = {{11}};
178 
179         int[][] map6 = {{55, 12, 25, 37, 78, 19, 81, 43, 83},
180                         {14, 23, 48, 59, 93, 52, 59, 96, 54},
181                         {72, 65, 19, 97, 35, 49, 41, 76, 13},
182                         {48, 65, 57, 44, 85, 24, 17, 15, 19},
183                         {79, 61, 65, 58, 61, 32, 72, 41, 18},
184                         {82, 39, 31, 53, 74, 58, 67, 32, 69},
185                         {15, 29, 28, 74, 99, 75, 17, 12, 56},
186                         {29, 33, 85, 11, 22, 67, 87, 94, 43},
187                         {91, 81, 95, 69, 76, 35, 16, 98, 21}};
188 
189         int[][] map7 = {{55, 21, 25, 37, 78, 19, 81, 43, 83},
190                         {14, 23, 48, 59, 93, 52, 59, 96, 54},
191                         {72, 65, 19, 97, 35, 49, 41, 76, 13},
192                         {48, 65, 57, 44, 85, 24, 17, 15, 19},
193                         {79, 61, 65, 58, 61, 32, 72, 41, 18},
194                         {82, 39, 31, 53, 74, 58, 67, 32, 69},
195                         {15, 29, 28, 74, 99, 75, 17, 12, 56},
196                         {29, 33, 85, 11, 22, 67, 87, 94, 43},
197                         {91, 81, 95, 69, 76, 35, 16, 98, 21}};
198 
199         int[][] map8 = {};
200 
201         int[][] map9 = {{33, 23, 13},
202                         {32, 21, 13},
203                         {22, 12, 31}};
204 
205         System.out.println(x: "Map 1");
206         printGrid(grid: map1);
207         path = findTreasure(array: map1);
208         System.out.println(x: path);
209 
210         System.out.println(x: "\n\nMap 2");
211         printGrid(grid: map2);
212         path = findTreasure(array: map2);
213         System.out.println(x: path);
214 
215         System.out.println(x: "\n\nMap 3");
216         printGrid(grid: map3);
217         path = findTreasure(array: map3);
218         System.out.println(x: path);
219 
220         System.out.println(x: "\n\nMap 4");
221         printGrid(grid: map4);
222         path = findTreasure(array: map4);
223         System.out.println(x: path);
224 
225         System.out.println(x: "\n\nMap 5");
226         printGrid(grid: map5);
227         path = findTreasure(array: map5);
228         System.out.println(x: path);
229 
230         System.out.println(x: "\n\nMap 6");
231         printGrid(grid: map6);
232         path = findTreasure(array: map6);
233         System.out.println(x: path);
234 
235         System.out.println(x: "\n\nMap 7");
236         printGrid(grid: map7);
237         path = findTreasure(array: map7);
238         System.out.println(x: path);
239 
240         System.out.println(x: "\n\nMap 8  null");
241         printGrid(grid: map8);
242         path = findTreasure(array: map8);
243         System.out.println(x: path);
244 
245         System.out.println(x: "\n\nMap 9");
246         printGrid(grid: map9);
247         path = findTreasure(array: map9);
248         System.out.println(x: path);
249     }//testTreasure
250 
251     /**
252      * Prints a 2-d array of 2 digit ints to standard output.
253      * Puts a space between each int.  Has column and row numbers.
254      *
255      * @param grid the array to print
256      */
257     public static void printGrid(int[][] grid){
258         if (grid.length == 0) //print nothing if empty
259             return;
260 
261         //top indices
262         System.out.print(s: "    ");
263         for(int col = 1; col <= grid[0].length; col++){
264             System.out.printf(format: "%2d ", args: col);
265         }
266         System.out.println();
267 
268         //border
269         System.out.print(s: "    ");
270         for(int col = 1; col <= grid[0].length; col++){
271             System.out.print(s: "---");
272         }
```

```
        System.out.println();

        //body and row numbers (with border)
        for(int row = 0; row < grid.length; row++){
            System.out.print((row + 1) + " |  ");
            for(int col = 0; col < grid[row].length; col++){
                System.out.printf( format: "%2d ", grid[row][col]);
            }
            System.out.println();
        }
    }//printGrid

    public static void testSumMinMax(){
        BSTree myEmployees = new BSTree();

        System.out.println( x: "Test sumMinMax empty tree");
        System.out.println( x: myEmployees.sumMinMax());

        myEmployees.insert( idNum: 50, name: "Bugs Bunny");
        System.out.println("Tree: " + myEmployees);
        System.out.println( x: myEmployees.sumMinMax());

        myEmployees.insert( idNum: 30, name: "Mickey Mouse");
        System.out.println("Tree: " + myEmployees);
        System.out.println( x: myEmployees.sumMinMax());

        myEmployees.insert( idNum: 80, name: "Minnie Mouse");
        System.out.println("Tree: " + myEmployees);
        System.out.println( x: myEmployees.sumMinMax());

        myEmployees.insert( idNum: 25, name: "Donald Duck");
        System.out.println("Tree: " + myEmployees);
        System.out.println( x: myEmployees.sumMinMax());

        myEmployees.insert( idNum: 65, name: "Pluto");
        System.out.println("Tree: " + myEmployees);
        System.out.println( x: myEmployees.sumMinMax());

        myEmployees.insert( idNum: 40, name: "Santa Claus");
        System.out.println("Tree: " + myEmployees);
        System.out.println( x: myEmployees.sumMinMax());

        myEmployees.insert( idNum: 100, name: "Queen Elsa");
        System.out.println("Tree: " + myEmployees);
        System.out.println( x: myEmployees.sumMinMax());

        myEmployees.insert( idNum: 70, name: "Anna");
        System.out.println("Tree: " + myEmployees);
        System.out.println( x: myEmployees.sumMinMax());

        myEmployees.insert( idNum: 58, name: "Olaf");
        System.out.println("Tree: " + myEmployees);
        System.out.println( x: myEmployees.sumMinMax());

        myEmployees.insert( idNum: 101, name: "Ausci 235");
        System.out.println("Tree: " + myEmployees);
        System.out.println( x: myEmployees.sumMinMax());

        myEmployees.insert( idNum: 10, name: "Java Reigns");
        System.out.println("Tree: " + myEmployees);
        System.out.println( x: myEmployees.sumMinMax());
    }

}
```