In this problem you will solve the “Rat in a maze problem” (Lecture 12), using Stacks and Queues.

The maze is given as a 2D array, where '0's are squares the rat can move to; '1's represent the squares where the rat cannot move to. The coordinates of the square at the top left corner are (0,0). The coordinates of the square at the bottom right corner are (14,12). The positive direction of x axis is to the right, and the positive direction of y axis is down.

You have to implement three methods:
searchStack(int fromX, int fromY, int toX, int toY),
searchQueue(int fromX, int fromY, int toX, int toY), and
searchStackSmart(int fromX, int fromY, int toX, int toY), which find a path from (fromX, fromY) to (toX, toY), simulating the actions of a rat using the following strategies:

**searchStack**: A rat that always attempts to move in a specific order: right, down, left, up. This method should be implemented using a stack.

**searchQueue**: On a given square, the rat reproduces, and up to four rats take one step away from the current square, each rat in a different direction. Each of these rats will choose one direction: right, down, left and up. This method should be implemented using a queue.
**searchStackSmart:** A rat that first attempts to move in the general direction of the exit, and moves away from the exit only when moving towards the exit has failed. For example, if the exit is down and right from the rat, the order of the first two directions it will attempt will be either “down, then right” or “right, then down”. This method should also be implemented using stacks.

The resulting output should be like this:

```
000100000010000
000100100000000
222222211111111
20011221001000
222001122100100
112201102100000
011201102100002
201221102100002
211021002222222
221220111111111
021210000000000
021210001000000
022210010000000
```

I’ve traveled through 78 square(s). The path contains 45 square(s)

where ‘2’s are squares on the resulting path. If there is a path available, your method should return true and your program should output the path it found. If there is no path available, your method should return false. You don’t have to find all the paths.

The distance the rat travels is equal to the number of stack (queue) pop actions your methods take during their execution.

The template for the class is given below:

```java
package dataStructures;

import java.util.Vector;

public class RatInMaze {
    int [][] maze_; 
    int width_; 
    int height_; 

    ArrayStack stack_; //You must use this variable to ref your stack
    ArrayQueue queue_; //You must use this variable to ref your queue
```
int traveled_;          //You should use this variable to store your pop actions
int pathLength_;        //You should use this variable to store the length of the path

public void load(Vector<String> maze)  //helper function
{
    height_ = maze.size();
    if(height_ == 0)
    {
        return;
    }
    width_ = maze.get(0).length();
    if( width_ == 0 )
    {
        return;
    }
    maze_ = new int[height_][width_];
    for(int i=0; i< height_; i++)
    {
        for( int j=0 ; j <width_ ; j++ )
        {
            maze_[i][j] = maze.get(i).charAt(j) - '0';
        }
    }
}

public void print(boolean found)              //helper function
{
    String line = "";
    String output = "";
    if(stack_ != null)
    {
        output = "I";
    }
    else
    {
        output = "We";
    }
    if (found) {
        for (int i = 0; i < height_; i++) {
            line = "";
            for (int j = 0; j < width_; j++) {
                if((maze_[i][j] == 1)||(maze_[i][j] == 2))
                {
                    line += maze_[i][j];
                }
                else{
                    line += 0;
                }
            }
            System.out.println(line);
        }
    }
}
public boolean searchStack(int fromX, int fromY, int toX, int toY) {
    stack_ = new ArrayStack();
    queue_ = null;
    //Your code goes here.
}

public boolean searchStackSmart(int fromX, int fromY, int toX, int toY) {
    stack_ = new ArrayStack();
    queue_ = null;
    //Your code goes here.
}

public boolean searchQueue(int fromX, int fromY, int toX, int toY) {
    stack_ = null;
    queue_ = new ArrayQueue();
    //Your code goes here.
}

// Please use the test code as follows.
public static void main(String[] args) {
    RatInMaze rim = new RatInMaze();
    Vector<String> maze = new Vector<String>();

    maze.add("000100000010000");
maze.add("000100100000000");
maze.add("000000011111111");
maze.add("000111001001000");
maze.add("000001100100100");
maze.add("110001100100000");
maze.add("011001100100000");
maze.add("001001100100000");
maze.add("011001000000000");
maze.add("001000111111111");
maze.add("001010000000000");
maze.add("001010001000000");
maze.add("000010010000000");

    System.out.println("\n\n");
    rim.load(maze);
    rim.print(rim.searchQueue(-1, 1, 10, 10));
    rim.load(maze);
rim.print(rim.searchStack(0, 0, 41, 1));

int fromX = 0;
int fromY = 7;
int toX = 14;
int toY = 6;
System.out.println("\n\n");
rim.load(maze);
System.out.println("A rat is searching:");
rim.print(rim.searchStack(fromX, fromY, toX, toY));

System.out.println("\n\n");
rim.load(maze);
System.out.println("Multiple rats are searching:");
rim.print(rim.searchQueue(fromX, fromY, toX, toY));

System.out.println("\n\n");
rim.load(maze);
System.out.println("A smart rat is searching:");
rim.print(rim.searchStackSmart(fromX, fromY, toX, toY));