Problem Solving Session

- The remainder of today's class will comprise the ***problem solving session*** (***PSS***).
- Your instructor will divide you into ***teams of 3 or 4 students***.
- Each team will ***work together*** to solve the following problems over the course of ***20-30 minutes***.
  - You may work on paper, a white board, or digitally as determined by your instructor.
  - You will submit your solution by pushing it to GitHub before the end of class.
- Your instructor will go over the solution before the end of class.
- If there is any time remaining, you will begin work on your homework assignment.

Class participation is a significant part of your grade (20%). This includes in class activities and the problem solving session.

Your Course Assistants will grade your participation by verifying that you pushed your solutions before the end of the class period each day.

Problem Solving Team Members

Record the name of each of your problem solving team members here.

Do not forget to ***add every team member's name***! Your instructor (or course assistant) may or may not use this to determine whether or not you participated in the problem solving session.

Luke Demi
Logan Nickerson

Problem Solving 1

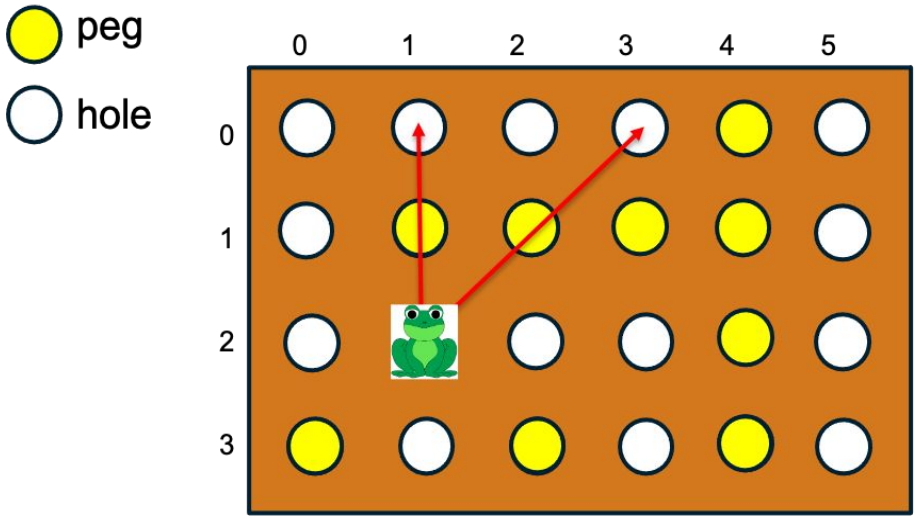
In the game of **HoppingFrog**, a rectangular board consists of **holes** and **pegs**. The frog is located at one of the holes and hops over an *adjacent* peg to another hole.

The diagram on the right shows the 2 possible locations to which the frog can jump.

The frog starting an initial location keeps hopping until it finally reaches a destination. The **path** used by the frog will be represented as a sequence of holes. A hole will be uniquely identified as its row and column coordinates.

While hopping, the frog can't visit the same hole that appeared in the path already, though it may hop the same peg more than once.

On the right, list all possible paths the frog may take starting from **(2, 1)** to reach **(0,5)**.



path 1: [(2, 1), (0, 3), (0, 5)]
path 2: [(2, 1), (0, 1), (2, 3), (0, 5)]
path 3: [(2, 1), (0, 3), (2, 5), (2, 3), (0, 5)]
path 4: [(2, 1), (0, 1), (2, 3), (0, 3), (0, 5)]
path 5: [(2, 1), (0, 1), (2, 3), (2, 5), (0, 3), (0, 5)]
path 6: [(2, 1), (0, 3), (2, 3), (0, 5)]

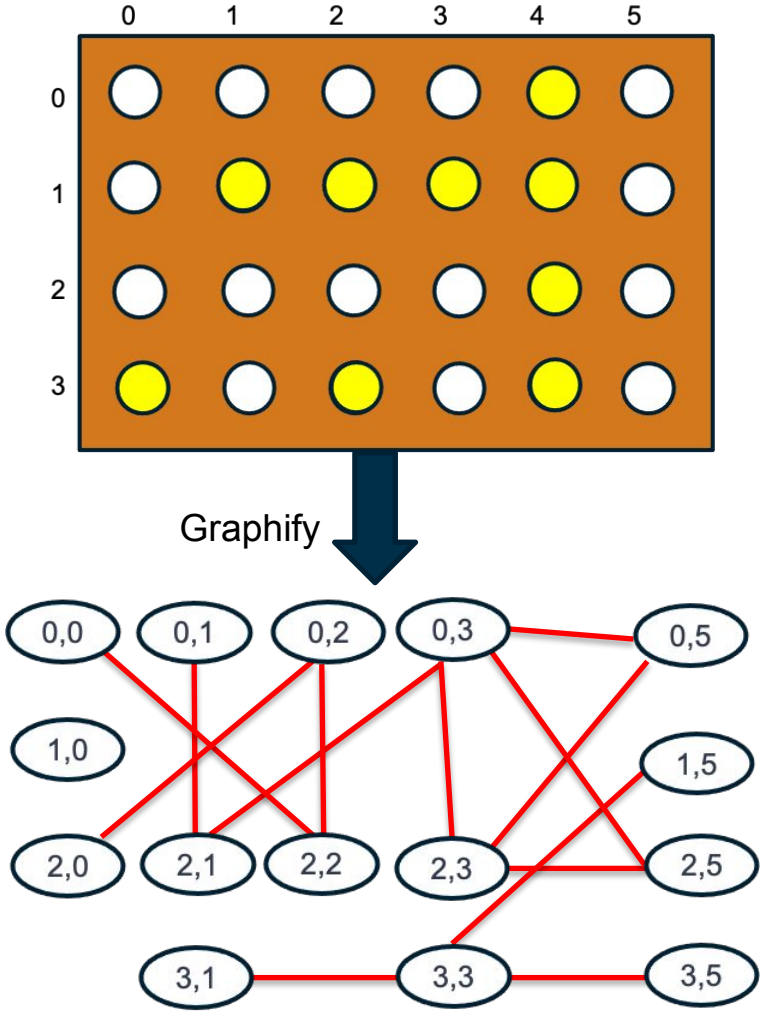
Problem Solving 2

We'd like to represent the board as an **adjacency graph** containing **holes** so that we can use DFS or BFS to find a path along which the frog can move.

Each vertex will contain a hole as data. Two vertices are connected if and only if there is only a single peg between the two holes.

The graph at right contains all of the vertices needed. Complete it by connecting them using

Can you now see how many connected components the graph has? 4



Problem Solving 3

Recall that a hole can be uniquely identified by its coordinates, row and column.

In the following code, instances of `Hole` (defined at right) are used as keys in a hash map.

```
Map<Hole, Integer> map = new HashMap<>();
map.put(new Hole(0, 0), 0);
map.put(new Hole(0, 3), 0);
map.put(new Hole(5, 2), 5);
System.out.println(map.containsKey(new Hole(5, 2)));
System.out.println(map.get(new Hole(5, 2)));
```

What will be the output of the code when it runs? Why?
True
5

Your program will use a `HashMap` to store holes. In order to make the map operations work as you expect, you need to override `hashCode` and `equals` in the `Hole` class.

Add the two special methods to the class at right. Make sure that two "equal" holes have the same hash code.

```
public class Hole{
    private final int row;
    private final int col;

    public Hole(int row, int col){
        this.row = row;
        this.col = col;
    }
    public int getRow(){
        return row;
    }
    public int getCol(){
        return col;
    }
        @Override
        public int hashCode() {
                return String.hash(row + ", " + col);
        }

        @Override
        public void equals(Object obj) {
                if(obj instanceOf Hole) {
                        Hole other = (Hole)obj;
                        return this.row == other.row && col ==
                        other.col;
                }
        }
```

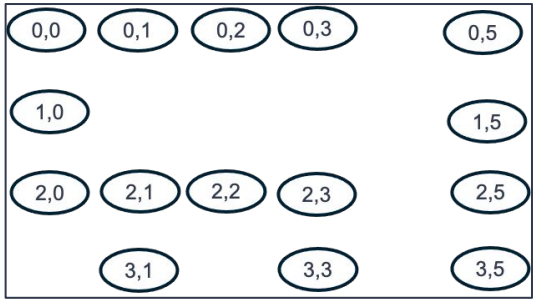
Problem Solving 4

Assume that you have been provided with a 2D `array` such as

```
char[][] array = {
    {'H', 'H', 'H', 'H', 'P', 'H'},
    {'H', 'P', 'P', 'P', 'P', 'H'},
    {'H', 'H', 'H', 'H', 'P', 'H'},
    {'P', 'H', 'P', 'H', 'P', 'H'}
}
```

which represents the game board and a `graph` containing all the vertices such as

`Graph<Hole> graph =`



How would you use `array` to connect vertices in `graph`?

In the space to the right, write your idea in Java. Do not assume that the dimension of array is 4x6.

```
public static void connectVertices(char[][] array, Graph<Hole>
graph){
        Graph<Hole> graph = new Graph<>();
        for(int i = 0; i < array.length; i++) {
                for(int j = 0; j < array[i].length; j++) {
                        if(array[i][j]
                        if(i + 2 < array.length) {
                                if(array[i][j]
                        }
                }
        }
```