In this assignment you will use a disjoint set forest to build a maze. Code is provided for all drawing operations so you have to focus on the maze-building part only. You will need to modify the maze.java program provided in the class web page. That program makes calls to the StdDraw.java library, which can be found at http://algs4.cs.princeton.edu/stdlib/StdDraw.java.html

We will designate the bottom-left cell as the starting point (the red circle in the figure) and the top-right cell (green circle) as the destination. Initially we will have walls separating all pairs of adjacent cells, forming a complete grid, then we will apply the algorithm based on disjoint set forests described below to remove randomly chosen walls, one at a time, until there is a unique path from the source to the destination.

We will use a disjoint set forest $S$ to indicate if a cell is reachable from another cell. Under this representation, if cell $i$ is reachable from cell $j$, $S.find(i) == S.find(j)$.

We’ll number the cells in an $N \times N$ maze 0 to $N^2 − 1$, as shown in the figure, thus cell 0 is the starting point, and cell $N^2 − 1$ is the destination. We will start with a complete grid, where all walls are present and no cell is reachable from any other cell. Then we will randomly choose pairs of neighboring cells, and, if these cells are not reachable from each other, we will remove the wall that separates them and update our disjoint set forest representation to reflect that. This process is repeated until all cells belong to a single set; at that point you display the maze. Having a single set guarantees that there is a unique path from the start to the destination and that every cell in the maze is reachable from the start cell (can you see why?).
The following pseudocode illustrates the process:

```
Create full maze with all walls present
Assign each cell to a different set in a disjoint set forest S
While S has more than one set
  Select a random cell C
  Select C', a random neighbor of C (Up or Right)
  If !S.inSameSet(C,C')
    S.union(C,C') //Remove wall that separates C and C'
Display maze
```

Implement the following four versions of the disjoint set forest and compare the total number of comparisons performed by each of them (to do this, use a global variable that will be incremented every time the *find* method is called). Run experiments with several different values of \( N \), including the largest size you can use before running into memory or time limitations.

2. Standard union with path compression.
3. Union by height.
4. Union by size.
5. Union by size with path compression.

As usual, write a report describing your work.