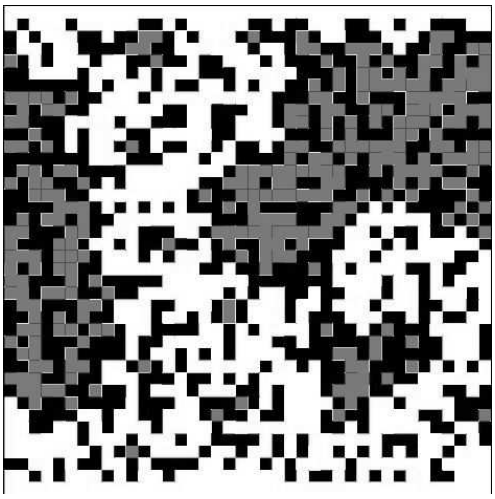
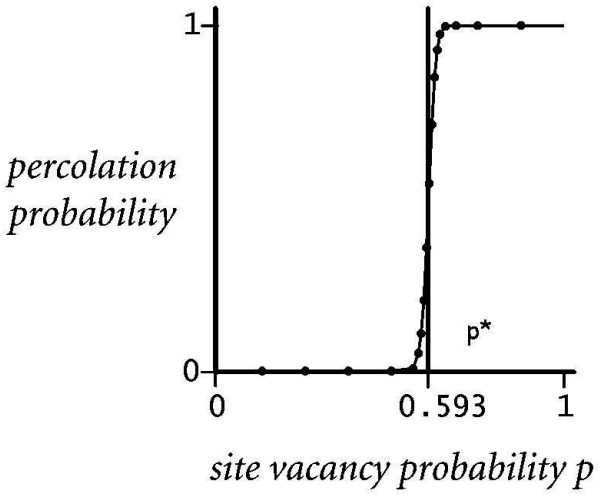
CS 312: Assignment 2

Out: Wednesday, Sep. 16th 2013
Due: Friday, Sep. 27th 2013

For this assignment, you will experimentally verify the percolation constant. You will need to answer the following question:

You are given a square grid of cells, of which a proportion *p* are vacant cells. Can you find a continuous path from top to bottom, visiting only vacant cells? For example, here is a 40x40 grid of cells: the black ones are filled, while the gray and white ones are vacant. Notice that there is continuous path of vacant cells from top to bottom. We say that this grid "percolates", because fluid poured at the top will find its way to the bottom.





The detailed question is, given a vacancy proportion *p*, what is the probability that the grid will percolate? Look at the experimental graph above. Notice that the probability of percolation rises sharply near *p=0.6*.

Approach

Run the following simulation, **many times**:

1. create an n×n grid of cells, initially all occupied.
2. *nvacant = 0*
3. repeat:
4. randomly make one more cell vacant
5. *nvacant++*
6. until the grid percolates
7. write down *p= nvacant/(n×n)*, the proportion of vacancies at which the grid percolated

If you run the simulation many times, you will obtain a robust estimate for the probability that the grid percolates, for each value of *p*.

Sub-problem 1: Randomizing an array

One problem you will run into is this: how do you pick a random cell, in order to make it vacant? You could generate two random numbers from 0 to *n-1* (assuming that random(n) returns a number from 0 to n-1):

- x = random(n)

- y = random(n)

and set *grid[x,y] = vacant*. However, that cell may already be vacant. If the grid already contains many vacant cells, you will probably not change the grid.

There is a better way.

First, generate the *[x,y]* coordinates for all cells in the grid, and put them in an array. Then, shuffle the array to randomize its order. Finally, visit the array from first to last, and you'll be visiting the cells in random order, without visiting any cell twice.

How do you shuffle an array? Try the Fisher-Yates shuffle

```
for (i = n-1; i >= 1; i--) {
  j = random number from 0 to i;
  swap(a[i], a[j]);
}
```

Sub-problem 2: Detecting percolation

Implement the union-find algorithm algorithm, as described in the textbook, and in the online lecture slides. Then, every time you make one cell vacant, add connections to its four neighbors, and check if the top row and the bottom row now belong the same connected component. If so, the grid percolates.

Thus detecting percolation requires you to test whether one of the n cells in the top row

- is vacant, and
- belongs to the same component as a vacant cell among the n cells in the bottom row.

This test costs $O(n^2)$ time. You can reduce the cost by adding two ficticious "plates" to your grid, one on top, and one on the bottom. Each cell in the top row is connected to the top plate, and each cell one in the bottom row is connected to the bottom plate (both plates are vacant). Then detecting percolation just requires testing whether the two plates are in the same component.

Deliverables

Submit the programs you used to calculate percolation, and a brief report which contains the numbers you obtained, and a graph of the percolation probability versus the occupancy fraction p.

Further work (in case you're bored)

What if the grid is hexagonal (or equivalently, is arranged in a staggered-brick pattern)? Now each cell has six neighbors, not four. Does this affect the critical p where percolation occurs?