Assignment a2

GiantBook

Results

The following table summarizes our results.
It shows the average number of random connections needed before the emergence of the giant component ("giant"), the disappearance of the last isolated individual ("no isolated"), and when the network becomes connected ("connected").

N	T	giant	(stddev)	no isolated	(stddev)	connected	(stddev)
100	100	74.37	6.9	258.63	62.13	259.47	61.34
1000	100	695.84	17.89	3747.1	618.17	3747.1	618.17
10^4	100	6,937.01	52.91	49,150.8	6,247.01	49,150.8	6,247.01
10^5	100	69,326.33	179.85	597,224.42	54,584.72	597,224.42	54,584.72
10^6	25	$6.93 \cdot 10^6$	548.67	$6.97 \cdot 10^6$	$5.17 \cdot 10^6$	$6.97 \cdot 10^6$	$5.17 \cdot 10^6$
10^7	10	$6.93 \cdot 10^7$	2,213.16	$8.47 \cdot 10^7$	$7 \cdot 10^7$	$8.47 \cdot 10^7$	$7 \cdot 10^7$

Our main findings are the following

The first thing we find, is that a giant component emerges after 695.84 random links (on average) for $N = 1000$. Secondly, we find that two of the events seem to happen simultaneously, namely the emergence of a *non-isolated* and *connected* network, which happens after 3747.1 (on average) for $N = 1000$.

Implementation details

We have based our union-find data type on WeightedQuickUnionUF.java from Sedgewick and Wayne: *Algorithms, 4th ed.,* Addison-Wesley (2011).
We added two fields of type *int* and *Set<Integer>* to keep track of the size of the largest component and the set of isolated individuals, respectively, at any given row in the experiment. We also added getter methods for these.

```
private int biggestComponentSize; // (M) the size of the current biggest component
private Set<Integer> isolatedIndividuals; // (M) set containing sites that are isolated

/**
 * (M) Return biggestComponentSize
 */
public int getBiggestComponentSize() {
    return biggestComponentSize;
}

/**
 * (M) Check if there exists isolated individuals
 */
public boolean noIndividualsIsIsolated() {
    return isolatedIndividuals.isEmpty();
}
```

Whenever the union() method is called, and the two component containing sites are merged, we set the value of *biggestComponentSise* to the size of the component from the biggest site.
The following bits of code are added to the conditional blocks that points the root of the smaller tree to the larger one, whenever the size of root *p* is bigger than the size of root *q* and otherwise.

```
// (M) Update biggestComponentSize
if (size[rootQ] > biggestComponentSize) biggestComponentSize = size[rootQ];

// (M) Update biggestComponentSize
if (size[rootP] > biggestComponentSize) biggestComponentSize = size[rootP];
```

Performance

WeightedQuickUnionUF initializes a data structure with *N* sites and takes linear time.
The Set *isolatedIndividuals* is given its values during this initialization.
The *union*, *find* and *connected* operations takes at most logarithmic time, and the logging of the size of the biggest component is implemented and executed within the *union* method.

We implemented Stopwatch.java from the algs4 library to measure the time spend to compute the solution for the emergence of giant component.
The average time was measured to

N	Time in seconds
100	0.000746
1000	0.001358
10^4	0.004257
10^5	0.01867
10^6	0.18171
10^7	0.30206

Using regression analysis on the data set above, we find that the power function

$$f(N) = 0.000033779 \cdot N^{0.57309}$$

approximately describes the time used to find the emergence of a giant component as a function of N.

Using this model, we can estimate that we would be able to compute the solution for

$$N = \sim 2.6 \cdot 10^{16}$$

if we let our algorithm run for 24 hours, assuming that we would never run out of memory or heap space.

Discussion

We defined the giant component to have size at least αN for $\alpha = \frac{1}{2}$

The choice of constant is important!

Changing the constant to $\alpha = \frac{1}{10}$ or $\alpha = \frac{9}{10}$ changes the experiment radically when observing the emergence for $N = 100$ compared to $N = 10{,}000{,}000$.