Giantbook Report

by Dennis Sadeler Shapira and Jonas Lomholdt

February 12, 2015

Results

The following table summarises 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	7.21×10^1	5.63	2.61×10^2	7.04×10^1	2.63×10^2	6.93×10^1
1000	10^6	6.96×10^2	1.76×10^1	3.74×10^3	6.40×10^2	3.75×10^3	6.39×10^2
10^4	10^5	6.93×10^3	5.53×10^1	4.90×10^4	6.43×10^3	4.90×10^4	6.43×10^3
10^5	10^4	6.93×10^4	1.75×10^2	6.04×10^5	6.47×10^4	6.04×10^5	6.47×10^4
10^6	1000	6.93×10^5	5.68×10^2	7.23×10^6	6.75×10^5	7.23×10^6	6.75×10^5
10^7	100	6.93×10^6	1.67×10^3	8.27×10^7	5.88×10^6	8.27×10^7	5.88×10^6

Our main findings are the following: The first thing that happens is that the giant component emerges, which happens at a time linear in N. Perhaps surprisingly, two of the events seem to happen simultaneously, namely that the last individual becomes non isolated and the network becomes connected, which happens at a time linear in N.

Implementation details

We have based our union–find data type on `WeightedQuickUnionUF.java` from Sedgewick and Wayne: *Algorihthms, 4th ed.*, Addison–Wesley (2011). We added two methods `getMaxComponentSize()` and `setMaxComponentSize()` by adding the following lines to `MyUnionFind.java`:

```java
public int getMaxComponentSize(){
    return maxComponentSize;
}

private void setMaxComponentSize(int size){
    if (size > maxComponentSize){ maxComponentSize = size; }
}
```

The `maxComponentSize` is a data field of type integer. This field keeps track of which component has the biggest size. It is updated in the `union` method, every time a union operation occurs. The `getMaxComponentSize()` method is self-explanatory. The `setMaxComponentSize` checks if the `size` parameter given is larger, than the last already seen max component size, and sets if that is the case. In the `find` method, we have added the line `parent[p] = parent[parent[p]];` to add the path compression feature. This is done to keep the tree structure flat by making the nodes point to it's grandparent each time the find method is invoked.

Assuming we never run out of memory or heap space, if we would let our algorithm for detecting the emergence of a giant component run for 24 hours, it could compute the answer for $N = 360\,000\,000\,000$.

We've run the code using a quick-find implementation as well. In 1 hour, we were able to handle and instance of size $N = 1.700.000$.

Discussion

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

The choice of constant is important; choosing $\alpha = \frac{1}{10}$ changes the experiment completely because the giant component can emerge way earlier than when half of the components must be connected as previously specified.