

Disjoint Sets 3 – Network Analysis (Giantbook)

Fri Feb 8 22:41:54 2019 +0100, rev. 053438e

Study the structure of social networks in the Erdős-Renyi model, such as the emergence of the giant component. This exercise serves as a very simple client for the Union-Find data structure.

Description

A *social network* consists of *individuals*, some of which are *linked*. You are welcome to think of the structure of “friends” defined by Facebook. In particular, the social networks studied in this exercise are symmetric in the sense that if p is connected to q then q is connected to p . (Other networks, such as the structure of “followers” on Twitter, are not symmetric.)

We need some standard terminology from network theory: An individual is *isolated* if it not link to any other individual. Two individuals u and v are *connected* if there is a sequence of individuals v_0, \dots, v_k with $v_0 = u$ and $v_k = v$ such that v_{i-1} and v_i are linked for each $i \in \{1, \dots, k\}$. A *connected component* is a maximal subset of connected individuals.

We are interested in the behaviour of networks as they grow when links are added at random. In particular, we want to know when following events occur:

nonisolated A network becomes *nonisolated* when it has no isolated individuals.

giant component A network has a *giant component* when it contains a connected component with at least half the individuals.

connected The network becomes connected when every individual is connected to all other individuals.

We study the dynamics of such a network in a very simple model closely related to what is known as the *Erdős-Renyi model*.¹ In this model, there are N individuals, all are isolated in the beginning, and links are formed completely at random. We want to know when the three events occur, and in which way the timing depends on N , the initial number of individuals.

Let me be very specific about the random model: In each round, select two individuals u and v uniformly at random. If $u \neq v$, then create a link between them. (It makes no difference if there was already a link between u and v before this round. Individuals are either linked or not linked.)

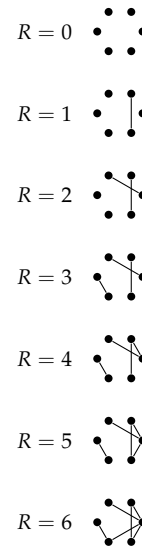


Figure 1: A tiny evolving network of 6 individuals. The *nonisolated* event happens at round $R = 3$, the giant component emerges at $T = 4$, and the whole network becomes connected at $R = 6$.

¹ There are actually two random models referred to as *Erdős-Renyi model*, and the one we describe and use here is somewhere in the middle. We chose this version because it leads to the simplest algorithms.

Deliverables

1. Your implementation of `GiantBook.java` or `GiantBook.py`.
2. A modified Union–Find data structure in a separate file. Call this `MyUnionFind.java` or `MyUnionFind.py`.
3. A project report in PDF, you are strongly encouraged to use the skeleton on the last page.

It is important that the functionality of the union–find data structure and of the client (`GiantBook.*`) are clearly separated. In particular, don't put everything in one file. Your extensions of the functionality of the abstract data type for union–find should be formulated in terms of meaningful, general methods that do not refer to the problem domain of the client.²

Suggested Roadmap

I suggest you focus on first implementing the functionality of the data structure correctly and sufficiently fast. (Only *afterwards* should you worry about performing random experiments and writing the report.) You start at a fundamental design choice: You can copy the source code of an appropriate version of the union find data structure from the books `algs4` library, and modify it to meet the requirements. Alternatively, you can write a 'wrapper' around such a data structure, i.e., a new data structure that has as the main ingredient an instance of the data structure from the book, and some additional information. The additional functionality is then achieved by—before and after calling the method of the original data structure—updating that extra information. Both approaches have their advantages and disadvantages, and this is completely your choice.

The preferred way of developing a functioning data structure is to expect input from standard input, as a text file. (That is much better than random input, because you can test, repeat, and debug the behaviour on the same text file.) Choose the same format as [SW, 1.5], such as `tinyUF.txt`, which is also what we did in the previous disjoint sets exercises.

Your first client should read the input, and determine the rounds in which the three events happen. The easiest event is *connected*, see the Tips below.

This is the algorithmically difficult part of this exercise. To help you test and debug correctness and performance, the Code Judge contains a battery of tests. You don't have to submit anything on Code Judge, but we hope the tests there are useful to make sure that your analysis tool is working as expected. The input format is the

² For instance, `boolean hasGiantComponent()` is not a good method for your union–find data type, because it needs to know how a giant component is defined in this particular application. On the other hand, `int maxComponentSize()` would be a perfectly fine method for a union–find data type that could be useful to many other clients as well.

following: in the first line a single number n giving the size, i.e. the universe is $0 \dots n - 1$. The following lines are pairs of elements that receive an link (this replaces the random source you should use). Observe that this input format is similar to the input for the previous disjoint sets exercises, just without the letter 'u' or 'q' because here all lines would be preceded by a 'u'. The output consists of 4 numbers in one line (separated by spaces), first n (redundant but useful for plotting the result), then (second) the first time the graph does not have isolated vertices anymore, then (third) the first time a giant component (with $\geq n/2$ vertices) formed, then (fourth) the first time the graph is connected. Here, time is the number of links that have been added for the event to occur, i.e., if we say that the graph of 10 nodes had a giant component for the first time at time 7 it means that we added precisely 7 links to create a component of 6 vertices.

Once this is done, you can run the experiments by feeding your implementation random links. There two schools of thought about this. You can either write another program that produces random test instances to standard output and pipes them into your GiantBook programme's standard input. This way, GiantBook remains untouched and another programme is responsible for random number generation, computing averages, etc. Or you can change GiantBook to perform the simulation itself. Both approaches have merit.

Tips

The simplest event to simulate is *connected*, because the Union-Find structure in the book already tells you if there is just one component. Make sure it works correctly for very small instances ($N = 10$) by following the computation step by step (print out what happens, check it by hand).

After that, detect the emergence of the giant component. This involves modifying the book's Union-Find implementation. I needed four lines of code.

Finally, check for nonisolated individuals. There is an obvious way of changing the Union-Find structure to detect the existence of isolated individuals (simply run through all elements and see if the size of their component is 1), but that will probably be too slow to be useful for large N . You need to do something that is both simpler and much faster. Again, I can do this in four lines of code.

Oh, and one more thing for the report: it's probably a good idea to print the values using scientific notation, i.e., like $6.96e+02$. You can use the `StdOut.printf` method with format string `"%.2e"` for that.

Questions

My java programme is slow. You should be able to detect the emergence of a giant component for $N = 10,000,000$ within a minute on a modern machine. If not, something is wrong. I am using weighted quick-union (with path compression).

My python programme is slow. You should be able to detect the emergence of a giant component for $N = 1000,000$ within a minute on a modern machine. If not, something is wrong. I am using weighted quick-union (with path compression).

What about those standard deviations? Reporting the result of random experiments using means and standard deviations is standard scientific practice. When performing experiments that involve randomness, the results will depend on the random choices. Therefore we can't just run your experiment once and report the result. Instead, the experimented is repeated T times, and we report the average (or "mean") of those experiments. So how large should T be? That's where the standard deviation comes in.³ This is not a course in statistics, so we won't make a big deal out of this, but your standard deviation should be an order of magnitude smaller than the mean value you report. For example, if you report mean 536, your standard deviation should be at most something like 63, but not 251. Otherwise you need to increase T . For small experiments (say, $N = 10$) this is not important. The standard library has methods for both computing both the mean and standard deviation of a sequence of values `StdStats`, so you don't write any code for this.

³ Read Wikipedia's entry on standard deviation for more.

Perspective

Want to learn more? You can read about the Erdős–Rényi model or more advanced models for social networks like the Watts–Strogatz model at Wikipedia. You can change your simulation to use that model instead, for example, and see what happens.

If you don't like the random graph models, check out the Stanford Network Analysis Project for real life data with large datasets. For example, running your code on the (famous) Enron email network should give the same value for the size of the largest component (called WCC in the dataset statistics table.)

Disjoint Sets 3 – Network Analysis – Report

by Alice Cooper and Bob Marley⁴

⁴ Complete the report by filling in your names and the parts marked [...]. Remove the sidenotes in your final hand-in.

Results

The following table summarises our results. It shows the average number of rounds needed before the emergence of the giant component (“giant”), the disappearance of the last isolated individual (“no isolated”), and when the network becomes connected (“connect”).

N	T	giant	(stddev)	no isolated	(stddev)	connected	(stddev)
100	100	7.15×10^1	5.77	2.68×10^2	6.32×10^1	2.69×10^2	6.3×10^1
1000							
10^4							
10^5							
10^6							
(10^7)							

Our main findings are the following: The first thing that happens is that [...] ⁵, which happens after a number of rounds [...] ⁶ in N . Perhaps surprisingly, two of the events seem to happen simultaneously, namely [...] and [...], which happen at after [...] rounds.

⁵ Replace by “the network becomes connected” or “the giant component emerges” or “the last individual becomes nonisolated”, whatever is correct.

⁶ Replace by “linear” or “logarithmic” or “quadratic” or something like that.

Implementation details

We have based our union–find data type on [...] .java from Sedgewick and Wayne: *Algorithms, 4th ed.*, Addison–Wesley (2011). We added a method [...] by adding the following lines to [...]:⁷

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 $N = [...]$.

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

⁷ And so on. Describe your modified union–find data type. You are encouraged to include code in this report; hopefully you added just a handful of lines and can explain everything needed to understand your changes in a single paragraph, maybe two. You don’t need to explain your *client* code at all.

Discussion

We defined the giant component to have size at least αN for $\alpha = \frac{1}{2}$. ⁸ The choice of α is arbitrary, choosing other constants (such as $\alpha = \frac{1}{10}$ or $\alpha = \frac{9}{10}$) gave essentially the same results. The choice of constant is important; choosing [...] changes the experiment completely because [...].

⁸ Remove and/or complete one of the following two sentences.