Designing Stimulating Programming Assignments for an Algorithms Course: A Collection of Exercises Based on Random Graphs

Michael Mitzenmacher *
Computer Science Division
U.C. Berkeley
Berkeley, CA 94720
mitzen@cs.berkeley.edu

Abstract

The field of random graphs contains many surprising and interesting results. Here we demonstrate how some of these results can be used to develop stimulating, open-ended exercises for courses in algorithms and data structures or graph theory. Specifically, we provide problems for algorithms that compute minimum spanning trees, connected components, maximum flows, and all-pairs shortest paths.

1 Introduction

We have found in teaching courses on algorithms and data structures that having students program some of the standard algorithms can be a useful learning experience. It ensures that they understand how the algorithms function, it provides them with experience in turning theoretical results into usable tools, and it demonstrates how theoretical time bounds translate (or fail to translate) into actual running times. We therefore often include programming in our assignments.

How do we evaluate the exercises we develop? We have several goals in mind that the exercises should accomplish. First, we hope to make the work interesting for the students. Second, we expect the exercises will teach the students something new, even beyond what they learn simply by the programming itself. Finally, we would like the exercises to tie in with the direction of the course, and perhaps with each other. These goals motivate us to develop exercises where the students do something beyond just writing the code and testing it on a few examples. Instead, we seek to create engaging and potentially open-ended questions that allow students to use their code to discover new and hopefully surprising things. In general, of course, such questions are hard to come by. However, we have found that the theory of random graphs provides a source of remarkable problems for several standard algorithms.

In this paper we demonstrate how the area of random graphs can be mined for engaging exercises by providing examples that students can explore after programming algorithms for the following problems: minimum weight spanning tree, connected components, maximum flow, and all-pairs shortest paths. Although stating the questions is easy, the related results in the literature are often rather technical, and are certainly beyond the scope of this article. We will not delve too deeply into these technical details, but instead merely attempt to provide basic intuition and references where possible.

Algorithms for all of the problems we consider can be found in [5, 9, 11], and many other standard texts. For more information on the area of random graphs, we recommend the indispensable *Random Graphs* by Béla Bollobás [2]. Throughout we assume that the students have simple and reliable pseudo-random number generators available in their programming environment. The question of how good a pseudo-random number generator should be is also beyond the scope of this paper; in practice, most systems provide reasonably good pseudo-random number generators.

*This work was supported in part by the Office of Naval Research and in part by NSF Grant CCR-9505448.

2 Minimum Spanning Tree

We begin by offering an example. We presume that the students have been asked to implement some standard minimum weight spanning tree algorithm.

Exercise: Consider a complete graph on n vertices. Assign a random weight uniformly and independently from [0,1] to each edge.

Plot the expected weight of a minimum spanning tree as a function of n. For a few values of n, plot the distribution of the weight for your samples. Do you notice anything? Can you make any conjectures? □

We have left open what ranges of n to consider and how many trials to make; these can be established by the instructor if desired. One might wish to encourage students to avoid floating point arithmetic, which can be slow on some systems. Instead assign a random integer weight from some large range (say [0,10000]) and then rescale.

One might expect that the weight of the minimum spanning tree grows linearly with n, since the tree contains $n-1$ edges. The actual result is rather peculiar: as n tends to infinity, the weight of the minimum spanning tree approaches a small constant! The following theorem, due to Frieze [6] and presented in [2, p. 141-144], is quite specific:

Theorem 1 *Let s_n be a random variable corresponding to the weight of the minimum spanning tree when the edges are given weights independently and uniformly from* $[0,1]$. *Then:*

$$\lim_{n\to\infty} \mathbf{E}[s_n] = \zeta(3) = \sum_{k=1}^{\infty} k^{-3} = 1.202\ldots$$

And, for every $\epsilon > 0$,

$$\lim_{n\to\infty} \mathbf{Pr}\{|s_n - \zeta(3)| \geq \epsilon\} = 0.$$

One way to think about the problem is the following: as there are $\binom{n}{2}$ edges, the lightest edge should weigh about $[\binom{n}{2}+1]^{-1} \approx \frac{2}{n(n-1)}$. Similarly, the next lightest edge weight should be about $\frac{4}{n(n-1)}$, and so on. Hence, if the minimum spanning tree consisted of the $n-1$ lightest edges, then its weight would be approximately 1. The minimum weight spanning tree does not generally consist of these edges, because they rarely form a tree; however, this result shows that this preliminary estimate is not too far off.

Although the theorem describes limiting behavior, students will find that the weight of the minimum weight spanning tree is concentrated near $\zeta(3)$ even for relatively small values of n. In Table 1 we present the average weight obtained from several simulation trials. Even at $n = 100$, the average derived from the simulation is very close to the limiting value. In Figure 1, we present an approximation for the distribution of the minimum spanning tree weight from 1000 trials for 100 vertex and 200 vertex graphs. (The approximation arises by making a histogram. Each data point covers a range of size 0.02.) Note that the mean of both curves is close to $\zeta(3)$, and moreover, the concentration around the mean is stronger for 200 vertices than for 100 vertices. This behavior is exactly what Theorem 1 suggests.

n	100	200	300	400	500	600	700	800	900	1000
s_n	1.207	1.201	1.200	1.206	1.202	1.205	1.205	1.199	1.204	1.201

Table 1: Average Minimum Spanning Tree Weight: 500 Trials

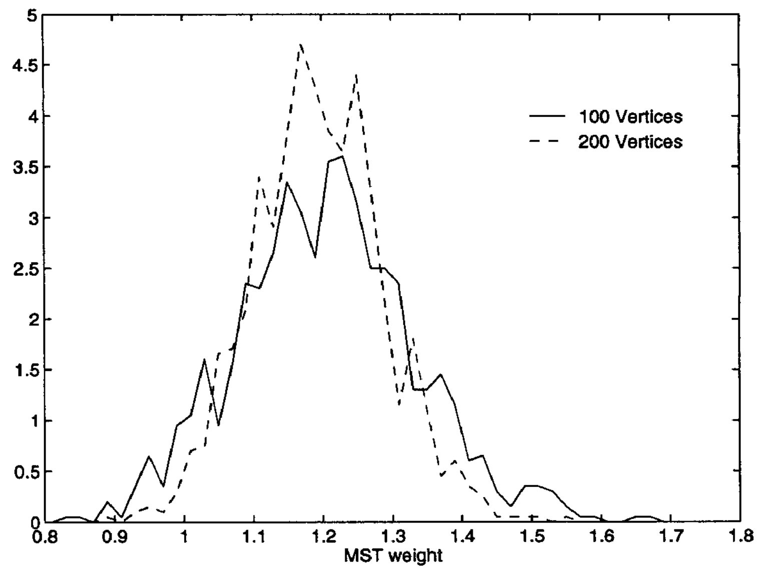

Figure 1: Minimum Spanning Tree Weight Density: 1000 Trials

We hope that students will find their simulation results surprising and worthy of further investigation. Making the exercise open-ended is simple, as there are a variety of issues the students can explore. For example:

1. What happens if we change the distribution of the weights assigned to the edges?
2. What happens if edges are first independently deleted from the graph with some probability q?
3. What is the distribution of the weight of the heaviest edge in the minimum spanning tree? The lightest edge not in the minimum spanning tree?
4. How many edges have weight less than the heaviest edge in the minimum spanning tree but are not in the minimum spanning tree?

Students can also be encouraged to develop their own questions to explore, or the exercise can be used to spark discussion on general probability questions, if the students have suitable backgrounds. For instance, it might be worth noting that the heaviest edge in a minimum spanning tree will generally weigh at least $\frac{\log n}{n}$, since the expected number of vertices whose adjacent edges all weigh at least this much is approximately 1.[1]

3 Union-Find: Connected components

Union-find refers to a class of algorithms used to maintain a data structure for disjoint sets, many of which have been analyzed by Tarjan. (For references, see [5, p. 461].) These algorithms are often used in connected components algorithms, which motivates the following exercise:

Exercise: Consider a graph that initially contains n vertices and no edges. Randomly include one edge at a time until the graph is connected, that is, until there is only one connected component.

Plot the expected number of edges that must be included before the graph is connected against n. Can you find a function $f(n)$ such that $f(n)$ is close to the expected number of edges that must be included? For a few values of n, use your samples to estimate $g_n(m) = \mathbf{Pr}\{\text{after } m \text{ edges, the } n \text{ vertex graph is connected}\}$. Plot your estimates of g_n; do you notice anything? $\square$

[1]All logarithms have base e unless otherwise specified.

The standard data structure for disjoint sets is particularly effective when one performs many operations that do not actually combine disjoint components, as may be the case under random edge insertion.

The number of edges that must be added before a graph becomes connected is very close to $\frac{n \log n}{2}$. In fact, $\frac{n \log n}{2}$ is a *threshold function*, in the following sense: as $n \to \infty$, if we throw in $\frac{n(\log n + \omega(n))}{2}$ edges, then the probability that the graph is connected approaches 1 if $\omega(n) \to \infty$ and it approaches 0 if $\omega(n) \to -\infty$. This follows from the following more specific theorem, paraphrased from [2, p. 150-151]:

Theorem 2 *If we throw in* $\frac{n(\log n + c)}{2}$ *edges, the probability that the graph is connected goes to* $e^{-e^{-c}}$ *as* $n \to \infty$.

In the terms of the exercise, the theorem says that $g_n\left(\frac{n(\log n+c)}{2}\right) \approx e^{-e^{-c}}$. Using this fact, we can get a good estimate for the expected number of edges required to connect the graph as n gets large. We approximate the probability density function and hence the expectation:

$$\begin{aligned} \mathbf{E}[\#\text{edges}] &\approx \int_{-\infty}^{\infty} \frac{n}{2}(\log n + c) e^{-c} e^{-e^{-c}} dc \\ &= \frac{n}{2}(\log n + \int_{-\infty}^{\infty} c e^{-c} e^{-e^{-c}} dc) \\ &\approx \frac{n(\log n + 0.577)}{2} \end{aligned}$$

This estimate proves remarkably accurate, even for small n, as can be seen in the results from a small set of simulations given in Table 2.

Vertices	Simulation	Estimate	Relative Error (%)
100	253	259	2.3
200	581	588	1.2
300	948	942	0.6
400	1314	1314	0.0
500	1685	1698	0.8
600	2076	2092	0.8
700	2440	2495	2.2
800	2874	2905	1.1
900	3331	3320	0.3
1000	3740	3742	0.1
1100	4146	4169	0.6
1200	4578	4600	0.5
1300	4991	5035	0.9
1400	5485	5474	0.2
1500	5927	5918	0.2
1600	6361	6364	0.0
1700	6842	6813	0.4
1800	7150	7265	1.6
1900	7673	7720	0.6
2000	8037	8177	1.7

Table 2: Simulation vs. Estimated Expected Number of Edges for Connectivity: 500 Trials

Students may get some idea of the threshold behavior by examining the distribution of results for a specific value of n. Figure 2 gives an estimated distribution for $g_{500}(m)$ based on a set of one thousand trials. (Each data point covers a range of 40 values.) The distribution is well concentrated around the mean. Threshold behavior often appears in the study of random graphs. For more details, see [2, pp. 37-38].

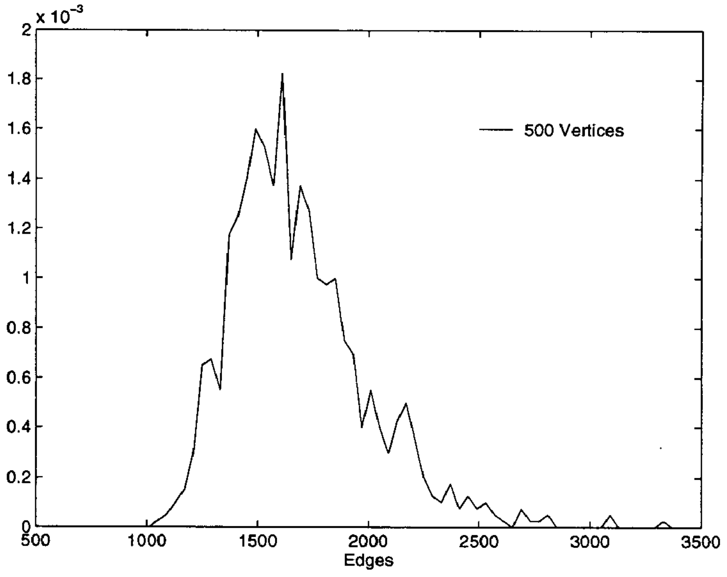

Figure 2: Estimated Density Function for #Edges for Connectedness: 1000 Trials

Again, this exercise raises many interesting questions, making it a good candidate for open-ended assignments. For example:

1. How many edges must be included before there is one large component, say with over half the vertices? Over some fraction p of the vertices?
2. What is the size of the largest component after m edges have been included? The second largest component?
3. What is the expected number of isolated vertices after m edges have been included?
4. What is the behavior on other types of graphs, such as lattices? (See, for example, [10].)
5. How well does the disjoint set data structure perform on this problem?

Students may be encouraged to experiment with these or other similar questions.

4 Flow algorithms: Bipartite matchings

Finding maximal bipartite matchings is a standard example of a use of maximum flow algorithms. We offer two interesting exercises based on bipartite matchings:

Exercise: Consider a bipartite graph that initially contains n vertices on each side and no edges. Randomly include one edge at a time until there is a perfect matching.

Plot the expected number of edges that must be included before the graph has a perfect matching against n. Can you find a function $f(n)$ such that $f(n)$ is close to the expected number of edges that must be included? Use your samples to estimate $g_n(m) = \mathbf{Pr}\{\text{after } m \text{ edges, the } 2n \text{ vertex graph has a perfect matching}\}$ for a few values of n. Plot your estimates of g_n; do you notice anything? □

Note that for the above exercise it is more efficient to update the matching as edges are introduced, rather than re-compute the maximum matching at each stage. This can easily be done using standard methods.

The number of edges that must be added before the graph contains a perfect matching is very close to $n \log n$. In fact, $n \log n$ is a threshold function, and if we throw $n(\log n + c)$ edges, the probability that the graph is connected goes to $e^{-2e^{-c}}$. See [2, p.73,155-159] for more details. Students should notice behaviors similar to those in the problem of when a graph becomes connected. Students may also explore some of the following questions:

1. How many edges must be included before there is a large matching, say with over half the vertices? Over some fraction p of the vertices?
2. Let $h_n(m)$ be the probability that there are no isolated vertices on the $2n$ vertex bipartite graph after m random edges have been added. What is the difference between $h_n(m)$ and $g_n(m)$?
3. Suppose one begins with k pairs of vertices already matched. How does this affect the number of edges that must be added before there is a perfect matching? Try both for small values of k, such as $k = 10$, and for larger values of k, such as $k = n^{2/3}$, over a range of values for n.

Exercise: Consider a bipartite graph with n vertices on each side. Suppose that for each vertex on the left we choose two vertices uniformly at random from the vertices on the right and include the corresponding edges. Let $f(n)$ be the expected size of a maximal matching. Experimentally determine how $f(n)$ grows. For a specific value of n use your samples to plot the an estimate of the probability that the maximal matching has at least m edges. Do you notice anything? □

This question was studied by Hajek [7]. As $n \to \infty$, $f(n) \approx 0.8381n$, and the larger n gets, the tighter the concentration of $f(n)$ about its mean. It is interesting to compare this result to the case where each vertex on the left chooses just one neighbor on the right, in which case the size of the maximum matching is approximately $n - n/e \approx 0.6321n$. The question can be expanded through a number of variations; we mention just a few:

1. What happens if each vertex on the left chooses 3 vertices at random? 4?
2. What happens if the left side contains βn vertices, for some constant β?
3. What happens if each vertex on the left and each vertex on the right chooses 2 random neighbors?

5 All-pairs shortest paths

Shortest paths algorithms are commonly used in network routing problems. Here we consider an interesting variation on a standard type of network, the hypercube.

Exercise: Consider a n-dimensional hypercube. Assign a random weight uniformly and independently from [0,1] to each edge. Let us call the length of the shortest path between two points the *distance* between them, and let us call the maximum distance between any two points on the hypercube the *diameter*. The diameter can be found by running an all-pairs shortest paths algorithm and finding the maximum shortest path between any two points.

Plot the expected value of the diameter as a function of n. For a few values of n plot the distribution of the longest path. Do you notice anything? Can you make any conjectures? □

At first, one might think that the diameter is linear in n, since there are 2^{n-1} pairs of points whose shortest paths must cross n edges. One can attempt to gain some intuition by considering paths chosen greedily between pairs of points. Take two points opposite each other on the hypercube, and at each step cross the smallest weight edge in a dimension that has not yet been crossed. In the first step, one has n dimensions to choose from, and hence the expected distance from the first step is $\frac{1}{n+1}$. At the second step, one chooses an edge from the remaining $n-1$ dimensions; the expected cost of this edge is $\frac{1}{n}$. Continuing in this manner, one finds that the expected cost of the greedy path between any two vertices is $O(\log n)$.

This intuition might lead one to suspect the correct answer is $O(\log n)$, but again, the actual result is dramatic: the diameter appears to be bounded by a constant, with high probability. In fact, we make the following conjecture:

Conjecture: The diameter of an n-dimensional hypercube with edge weights independently and uniformly distributed from $[0,1]$ is at most 2 with high probability for all n.

We gave this exercise on an assignment, and were surprised by the results. The following intuition seems to be correct[2]: if one looks only at edges with weight at most c_1/n for some constant $c_1 > 1$, with high probability the remaining subgraph contains a large component, consisting of a constant fraction of the vertices. Every pair of points can be connected by a path that lies primarily on this large component; that is, there is path that uses at most $O(n)$ edges on this component and constant number of other edges. Hence the maximum shortest path will be of constant length. This particular problem seems not to have been mentioned previously in the literature, although several related results can be found. (For example, see [1, 3, 4, 8].) Proving tight bounds on the diameter appears difficult, and we would be interested in a proof.

Because the number of vertices and edges grows exponentially with the dimension, students should not be expected to run this experiment for large values of n. In Table 3 we offer some data from our simulations on up to ten dimensions. Additional questions one might ask include:

1. How often is the shortest path between two points a greedy path?
2. What does the longest path look like?
3. What happens if we change the distribution of the edge weights?

Dimension	2	3	4	5	6	7	8	9	10
Diameter	0.878	1.213	1.445	1.627	1.723	1.794	1.822	1.849	1.865

Table 3: Average Hypercube Diameter: 500 Trials

6 Conclusion

We hope that the specific exercises we have suggested here prove useful. The goal is to make programming algorithms more interesting for students by demonstrating surprising and unusual results. Perhaps the exercises will encourage students to explore related questions, or spark them to learn more about graph theory or random graphs.

We believe that there are many other similar potential exercises out there, some based on results in the literature and some as yet undiscovered. We would be interested to hear if others design their own such exercises. Random graphs are a wonderful area for exploration, especially since even simple algorithms can be used to experiment with difficult problems.

7 Acknowledgments

The author would like to thank several people for interesting discussions or for reading early drafts of the article, including Andrei Broder, Michael Clancy, Steve Lumetta, Lars Rasmussen, Alistair Sinclair, Umesh Vazirani, and David Wolfe.

[2]Thanks to Andrei Broder for suggesting this argument.

References

[1] M. Atjai, J. Komlós, and E. Szemedrédi. Largest Random Component of a k-Cube. *Combinatorica.* **2**, 1982, 1-7.

[2] B. Bollobás. *Random Graphs.* Academic Press, 1985.

[3] B. Bollobás, Y. Kohayakawa, and T. Łuczak. The Evolution of Random Subgraphs of the Cube. *Random Structures and Algorithms.* **3**, 1992, 55-90.

[4] B. Bollobás, Y. Kohayakawa, and T. Łuczak. On the Diameter and Radius of Random Subgraphs of the Cube. *Random Structures and Algorithms.* **5**, 1994, 627-648.

[5] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press, 1992.

[6] A. Frieze. On the Value of a Random Minimum Spanning Tree Problem. *Discrete Applied Math.* **10**, 1985, 47-56.

[7] B. Hajek. Asymptotic Analysis of an Assignment Problem Arising in a Distributed Communications Protocol. *Proceedings of the 27th Conference on Decision and Control.* 1988, 1455-1459.

[8] A.V. Kostochka, A.A. Sapozhenko, and K. Weber. Radius and Diameter of Random Subgraphs of the Hypercube. *Random Structures and Algorithms.* 4, 1993, 215-229.

[9] D. Kozen. *The Design and Analysis of Algorithms.* Springer-Verlag, 1992.

[10] S. Lumetta, A. Krishnamurthy, and D. Culler. Towards Modeling the Performance of a Fast Connected Components Algorithm on Parallel Machines. Available at http://www.cs.berkeley.edu/~stevel. Conference version at http://www.supercomp.org/sc95/proceedings/465_SLUM/SC95.HTM.

[11] R. Sedgewick. *Algorithms.* Addison-Wesley Publishing Co., 1988.

*************************************C++ Arrays From Page 28*************************************

Borland C++ Version 3.1 Library Reference. Borland International, Inc. Scotts Valley, California. 1992.

Horowitz, E., Sahni, S., Mehta, D.. Fundamentals of Data Structures in C++. Computer Science Press. New York. 1995.

Pohl, Ira. C++ for Pascal Programmers Second Edition. The Benjamin/Cummings Publishing Company, Inc. Redwood City, California. 1994.