

On the Science Behind Computing (Part I)

Samir Khuller *

1 Graphs and Facebook

Facebook is a social networking website. From my point of view it is simply a graph – people correspond to nodes in this large graph, and edges correspond to friendship links (neighbors). Let us consider a very basic computational problem faced in facebook. One of the things that facebook does is to make “friend suggestions” - these may be based on several factors, however once you join the network, one possible rule that facebook might use is to consider pairs of individuals that have common friends, and then examine their data and interests and make recommendations for a subset of these pairs.

Lets say I join facebook, and invite 20 other people to become my friends in facebook. If 18 accept my invitation, then I have 18 friends - my degree in this graph is 18. However, facebook might observe that I am friends with Kristin, and Jessica is also Kristin’s friend, and might make a “friend suggestion” to me about Jessica since we have a common friend (Kristin). (In practice, facebook may use other parameters as well such as only suggesting people who have multiple friends in common, or might look at other data as recommending friends who attended the same college and have a friend in common.)

How can we determine which pairs of individuals who are currently not friends, have a friend in common?

This is a fairly simple computational problem, but as we will see it also leads to an interesting discussion about representing the facebook graph as well as the issues that are relevant while designing algorithms. After we produce a list of such individuals, we might want to further analyse this list to decide which ones to actually recommend.

We will assume that the graph has N nodes in all. Current estimates put N in the range of several million users. Lets assume for simplicity that we have ten million users¹ ; this suffices to make our main point. Most users will have a much smaller list of friends - lets assume that we bound the maximum number of friends a user can have by Δ (this in effect is an upper bound on the degree of any node in the graph). Let say that we set this bound to 1000; this seems like a reasonable number, (except for those few insanely popular people).

Let d_i be the *degree* of node i . Let V be the set of nodes and E be the set of edges.

We first make some simple observations about the graph itself.

$$2|E| = \sum_{i \in V} d_i$$

This essentially says that if you add the degrees of all the nodes in the graph you get exactly twice the number of edges. (Please draw a few graphs to convince yourself of this.)

*Department of Computer Science, University of Maryland, College Park, MD 20742. E-mail : samir@cs.umd.edu.

¹In class we assumed one million users but this is a better estimate.

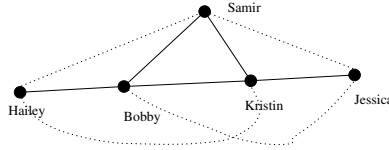


Figure 1: Figure showing a small social network. Pairs of nodes that have a friend in common are also shown.

For an arbitrary graph since the degree of any node is at most $N - 1$, we can easily show that $2|E| \leq N(N - 1)$ by simply saying that $d_i \leq N - 1$ and adding the N terms gives us a bound of $N(N - 1)$.

Also note that $d_i \leq \Delta$ by our assumption on the facebook graph. Thus for this graph we can replace the upper bound of $N - 1$ by Δ and show that $2|E| \leq N \times \Delta$.

Let us now focus on our original problem of finding all pairs of nodes with the property that they are not friends, but have at least one common friend. Lets try the following algorithm, we call it the “Generate Pairs” algorithm.

First make a list of *all* possible pairs of nodes. From this list we shall eliminate the pairs of nodes that are already friends. Suppose our list of all possible pairs (see Fig. 1) is $\{(Hailey, Bobby), (Hailey, Samir), (Hailey, Kristin), (Hailey, Jessica), (Bobby, Samir), (Bobby, Kristin), (Bobby, Jessica), (Samir, Kristin), (Samir, Jessica), (Kristin, Jessica)\}$. We first eliminate the five pairs that are already friends and get the list $\{(Hailey, Samir), (Hailey, Kristin), (Hailey, Jessica), (Samir, Jessica), (Bobby, Jessica)\}$.

For each pair on this list we need to check if they have a common neighbor. In other words, if we consider the pair (Samir, Jessica) we need to examine all the neighbors of Samir and all the neighbors of Jessica and determine if the two lists have a common element. So if the graph does have the edge (Jessica, Kristin) and the edge (Kristin, Samir) then we will know that the two do indeed have a common friend (there may be others as well). Note that Hailey and Jessica do not have any common friends. If there is no common friend we can remove the pair from the list. This check is not trivial and may take time proportional to the degree of a node d_i . To derive a crude upper bound on the running time of the algorithm let us assume that we need to do the following. For each pair of nodes (X, Y) we have to go through all the neighbors of X , and for each such neighbor check to see if any of them are neighbors of Y . This may take upto Δ steps (since we have to check go over all the neighbors of a node).

After removing the existing friends from the list, the list of pairs can still be quite large. In general we will have $\frac{N(N-1)}{2} - |E| \geq \frac{N}{2}(N - 1 - \Delta)$ such pairs on our list. (We discussed this point earlier – this was the number of handshakes when N people were at a party and everyone shook hands with everyone else. We subtracted $|E|$, the number of edges since we only want to make a list of the possible pairs that *do not* have an edge in common and we know that $2|E| \leq N \times \Delta$, so by subtracting an even larger quantity $(N \times \Delta)$ we get a smaller number.) However for each such pair we are spending “roughly” (this is a slight over count clearly) Δ steps to check if they have a common neighbor. The total number of steps is therefore roughly $\Delta \frac{N}{2}(N - 1 - \Delta)$. If $\Delta = 10^3$ and $N = 10^7$ then we get a rough estimate of 5×10^{16} . This is a HUGE number! Even if a computer performs 10^9 operations per second, this will take 5×10^7 seconds. A day has 86,400 seconds. This is almost 600 days! (In class we got an answer that was around 10 days, but that was for a million node graph.)

Let us consider a slightly different method next. We call this the “Node Centric” method.

Our focus will not be to generate a list of pairs of non-adjacent nodes but simply to focus on an individual X . This person has at most Δ friends. Suppose we consider all the friends of friends - how large can that set be? Observe that this set has size at most Δ^2 (try a simple example to check this). By definition, any such node Y in this set already has a common neighbor with X . All we need to check for each such Y is that if they already have an edge to X or not. As long as we can do this quickly, we are in business and can generate all the potential pairs involving X in Δ^2 computational steps. However, we still have to repeat this for every possible choice X and this increases the computation by a factor of N . In any case this method takes no more than $N\Delta^2$ steps, which is $10^7 \times (10^3)^2$ or 10^{13} . If we can perform 10^9 computations per second, then it takes 10^4 seconds compared to 600 days by the previous method. This is less than 3 hours of computation time. Note that we were able to achieve this by a simple change of the algorithm. Can we do any better?

We will next discuss graphs and their representation. Clearly we cannot represent a million node graph by drawing all the links.