# The Algorithm

We construct by taking and adding a new node on every edge. For example, say we have edge from , and we add a new node on it for , it would become two edges and .

So is constructed as such.

is .

is (all the original nodes/people from ).

# Proof of Correctness

**Claim**: if has an independent set of size , then has a very independent subset of

the set of size .

**Proof**: We can rephrase this claim to *if* *has an independent set of size , then has a very independent subset of the set of size at least* , because if it has a subset of with more than elements, say it has elements, then just take away any nodes, as the remaining nodes would still be very independent. can’t have less than very independent people (nodes) because we’ve separated every old node from each other with a new node, so all the original independent people who shared a mutual friend no longer do, and will be classified as very independent. And of course, all the original independent people who didn’t share mutual friends would still be correctly classified as very independent. In short, we don’t “lose” independent people in my friend’s code.

**Claim**: if has no independent set of size then also doesn’t have a very

independent subset of the set of size .

**Proof**: We rephrased the previous claim to *if* *has an independent set of size , then has a very independent subset of the set of size at least* . The contrapositive of a statement has the same truth value of the original statement, so the contrapositive of that claim is also true: *If G’ doesn’t have a very independent subset of of size at least k, then doesn’t have an independent set of size .* Notice that the hypothesis in this conditional statement is the same as the hypothesis in the current claim we’re trying to prove, so we just need to prove a specific case of the conclusion of that contrapositive statement, that wouldn’t have an independent set of size at least . You can’t have more independent people than very independent people from running my friend’s because

# Runtime Analysis

The reduction algorithm (computing , , and ) is linear in the number of edges because we go to every edge in and just add a new node, which effectively is deleting that old edge and adding two new edges. Getting and is constant because we’re just equating them to old inputs. If we let , this algorithm is .