Checkpoint 2

Overview of developments

- Massaged the definition of BFS to allow using functional induction.
- Broke the definition of correctness down to separate properties.
- Wrote down a plan for how to write proofs for different properties of BFS.
- Identified loop invariants for the major goals (unlikely that all are final).
- Proved intermediate lemmas for "BFS only outputs edges that are in the graph".
- Cleaned up the silly things due to using Qed instead of Defined.

Details

The main issue was getting functional induction (or anything like that to work). Our definition of BFS closely follows how one would write the algorithm in an imperative language, and the structure of the proof does not always follow the structure of any single argument / loop variable. Rather, for example, the invariant might be "each node that is reachable from the start in the original graph is reachable from the frontier in the residual graph". Therefore, we need to be able to conveniently reason about how one iteration changes the state. Initially, we implemented BFS using Function, but it did not give us functional induction. Then we implemented it using fuel and proved that length(nodes(graph)) is sufficient fuel, but doing induction on that was also rather cumbersome. We tried to define our own induction principle for the version using Function, but after a while of fiddling around with the definition functional induction started working. We are not sure what exactly happened, and we did not commit after every small change, but it seems that refactoring the right amount of one iteration into a separate function made functional induction work.

We reworked the definition of correctness of BFS. It is now broken up into several smaller lemmas that can be proven separately. In addition it fixes bugs with the previous definition. For each lemma, we stated a loop invariant which we will try to use to prove it (if applicable) and a (probably incomplete) list of helper lemmas. The general strategies for the major goals look like the following:

- `finds_legit_paths`: the parent array represents only paths that exist in the graph and start from a frontier node
- `finds_the_shortest_path`: for all nodes, if there exists a path from the node to one of the frontier nodes shorter than the path that would currently be created, then the parent array contains it. Note that the base case is trivial, since nothing shorter than length 0 exists.
- Proving that BFS finds paths to all reachable nodes is currently our weakest point. The general argiment we have is "if there is an edge from a node

that bfs finds (for example, the start node)", to another node (so it is reachable), then bfs at the start node of that edge will find the following node.

When the technical issues (induction, Qed vs Defined) were worked out, we got started proving the things we knew we would need anyway that we weren't able to prove before. It was mostly just "videogame." and everything has so for gone as expected.

Next steps

- There are a bunch of lemmas for which we have relatively precise proofs in our heads that just need to be written down. We hope that some of them will "just work" as many things started doing after getting proper functional induction.
- Organize the repetitive "destruct; destruc; .. injection" sequences into pattern-matching-based patterns. It is probably a good idea to wait for a couple of more lemmas before doing this, but I expect the size of the existing proofs to shrink a lot when this is done.
- Figure out whether BFS returning a parent array instead of a list of paths is a good idea or not. It is closer to our own model of how BFS works (which originates from contest programming and is thus performance-oriented), but the high-level lemmas we state often talk about the extracted paths. It might also just work either way, in which case we will keep it as it is – this is how we would write the code if it wasn't for Coq, so the proof will be more meaningful.
- Big open-ended question: how much should we be going top-down and how much bottom-up?