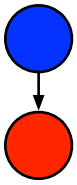
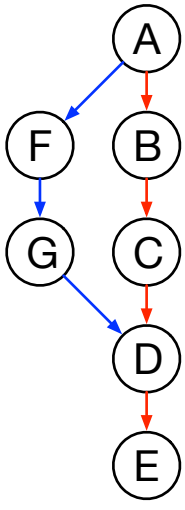


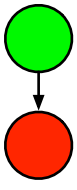
The blue commit depends on the red commit.



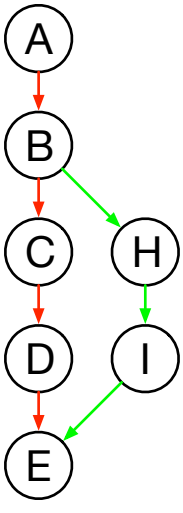
A file graph consists of nodes and edges. Each node contains a line of text (or a chunk of binary data). Each commit adds edges, and optionally nodes. In this case the red commit created the file ‘ABCDE’, and the blue commit replaced ‘BC’ with ‘FG’.

The *cogent path* is the path that begins at the src node and ends at the snk node and traverses all edges from each commit X unless another commit Y, which depends on X, has routed the cogent path around those edges.

If the cogent path exists then the nodes in the cogent path contain the data of the file and can be found trivially by always taking the outgoing edge that came from the most dependent commit.

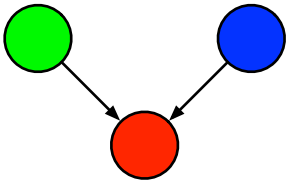


The green commit depends on the red commit.

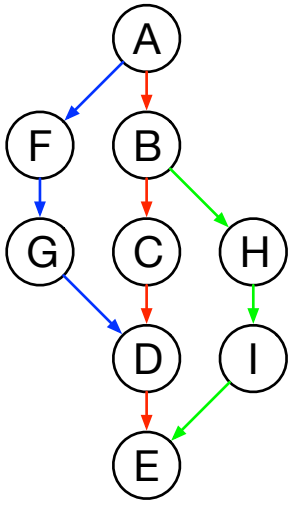


This is an example of a different commit applied to the same initial commit. This could have been a commit created by a different contributor that hasn’t tried to merge the blue commit into her repository.

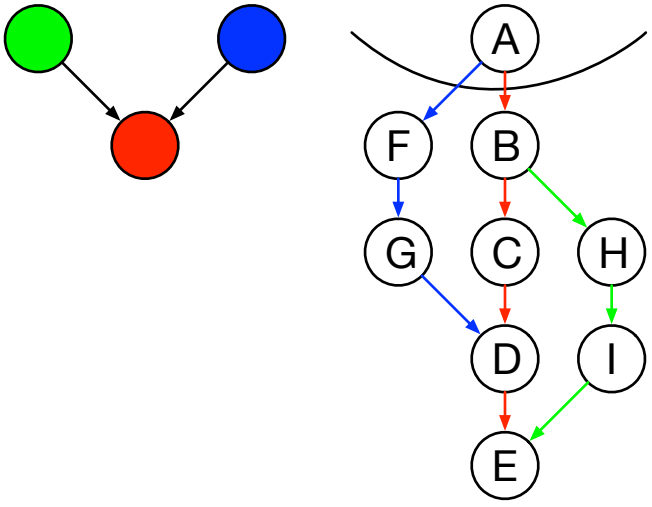
The cogent path here is ‘ABHIE’.



The blue and green commits are independent of each other, but both depend on the red commit.

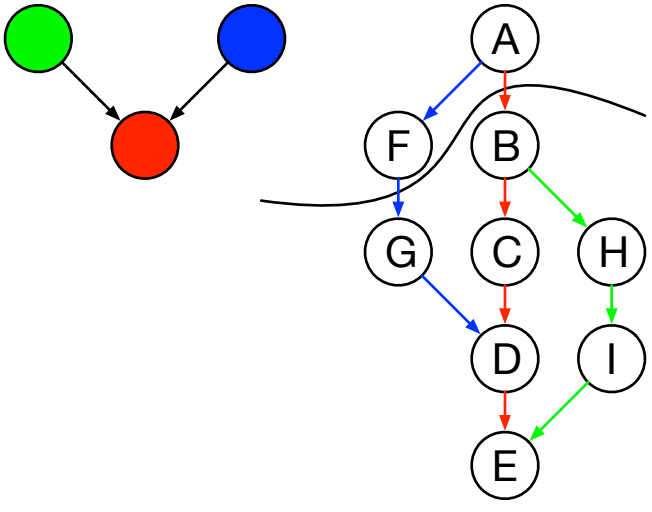


When these three commits are all merged together we find that there is no cogent path. Blindly following edges from the highest commit in the dependency graph will yield ‘AFGDE’, but it ignores the data from the green commit. Logically this must be a conflict, since data from the green commit has been deleted implicitly, and all modifications must be explicit, but we need a better way to define and detect conflicts.

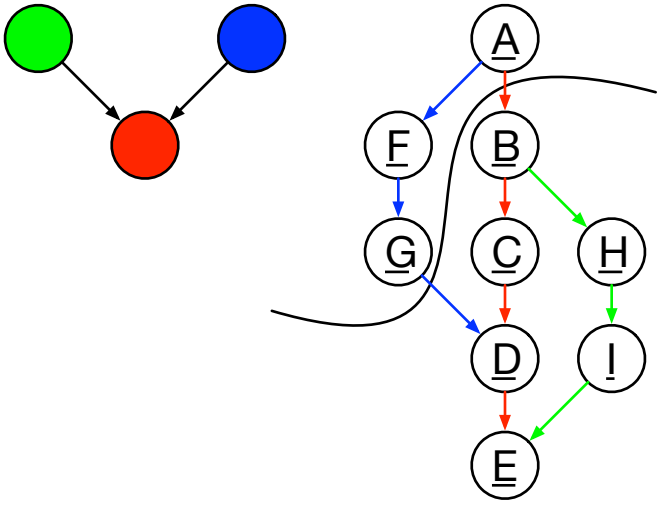


Let's define a cut through the graph called the *verge*. The verge starts containing no nodes and we advance it down the file one node at a time, only ever moving past a node if the verge has already passed all of that node's predecessors.

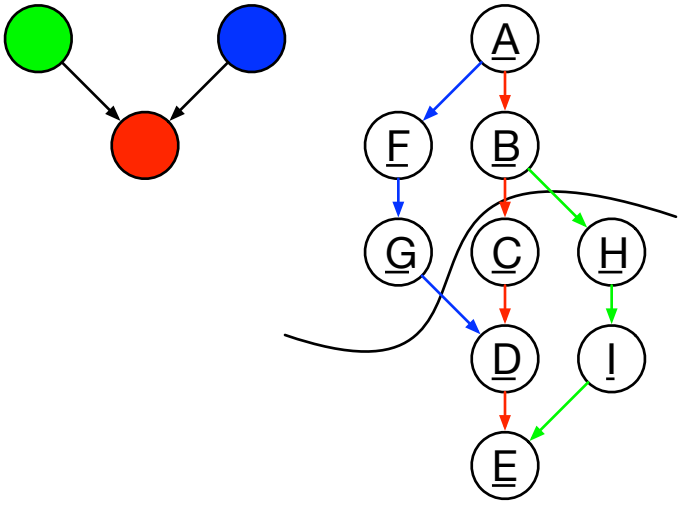
Given any position the verge is at we can always determine the next node rendered in the file, it will be the one we find by following the least dependent edge currently cut by the verge. If the cogent path exists then, by definition, this edge is on the cogent path. In this case the verge has passed A, and since the blue commit depends on the red commit, the next edge in the cogent path, if it exists, will be the edge from A to F.



Here we've advanced the verge by moving it past F. Alternatively it could have been moved past B. We can see that the next edge in the cogent path, if it exists, will be the edge from F to G.

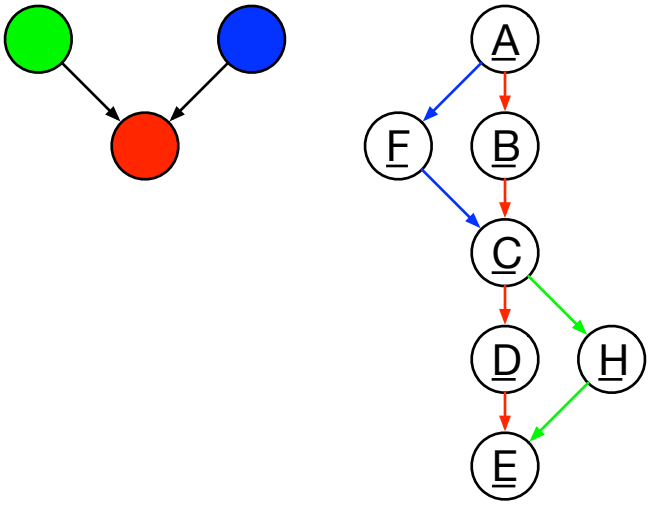


Again the verge has advanced, and again we could have chosen to advance past B but chose G instead.

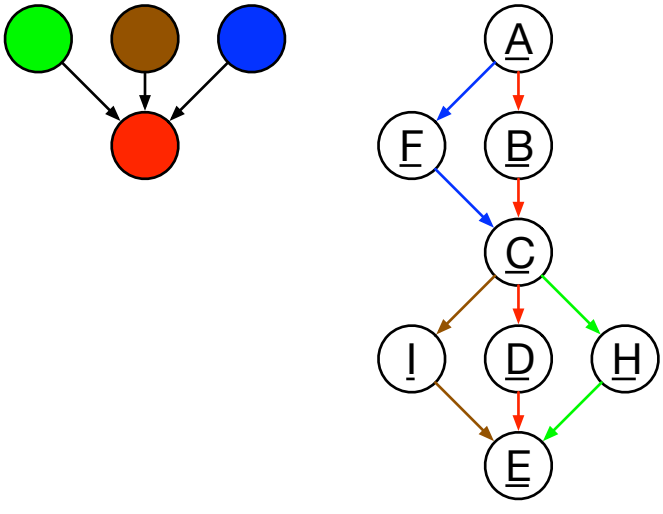


Things get interesting! We couldn't advance the verge past D because we hadn't also moved it past C, so we were forced to move it past B. Now the verge is cutting three edges, one from each commit. There is no way to decide between blue and green since neither depends on the other, so the cogent path does not exist and this file is in conflict.

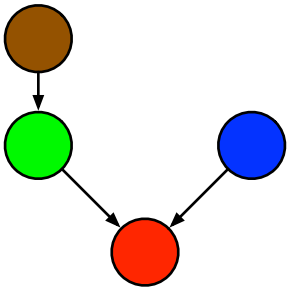
.

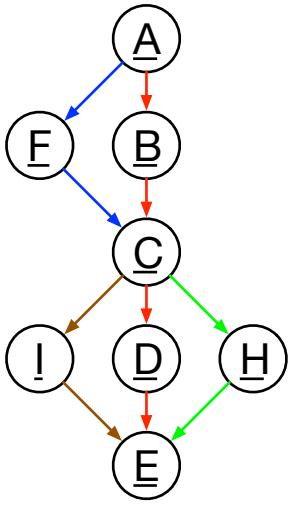


Here is an example with two independent commits that don't conflict. There is no way to advance the verge such that it cuts a green and blue edge at the same time. Traversing the graph by always taking the edges from the most dependent commit yields 'AFCHE'.

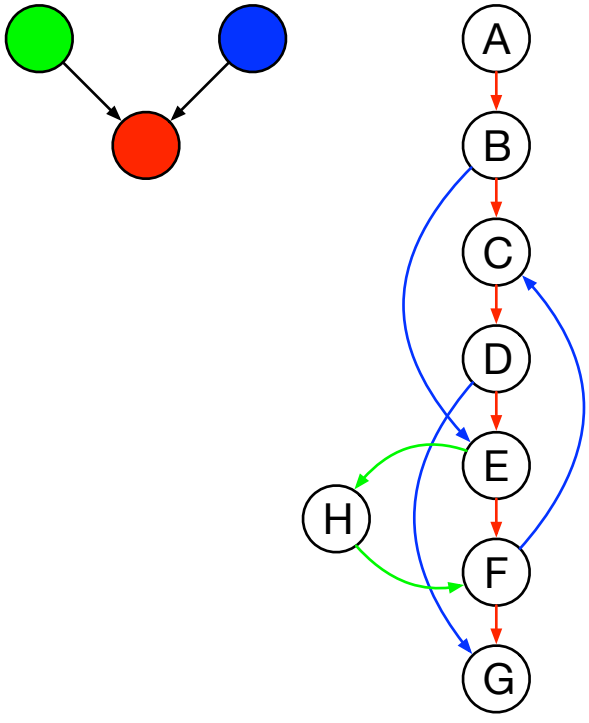


Here we've added an additional commit onto the graph above. Now we cannot advance the verge through the entire file without cutting two independent edges: the edge from C to I and the edge from C to H. The cogent path can't exist because following the brown edges would avoid the green ones and vice versa.





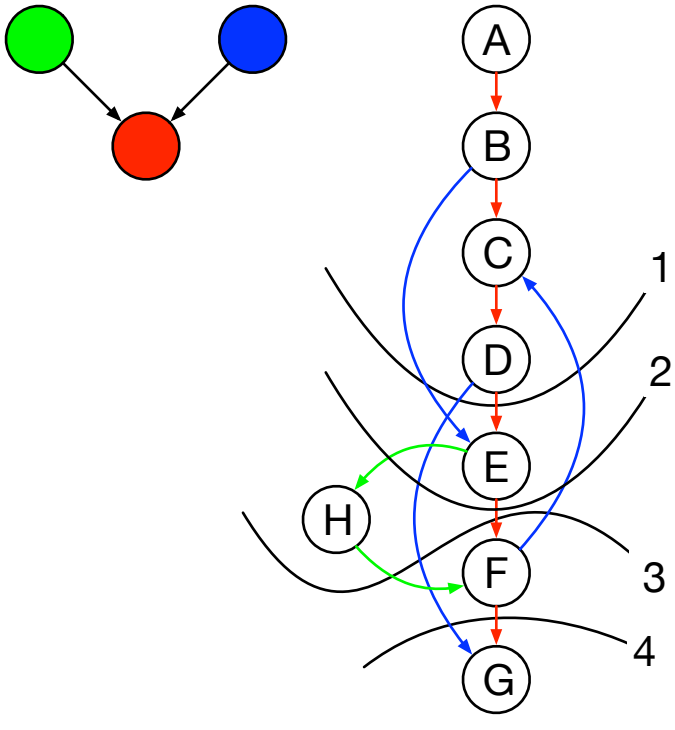
This is like the above example except that the brown commit has declared a dependency on the green commit. This means that the verge can't cut multiple independent edges, so the cogent path exists and the file reads 'AFCIE'.



This is an example of a move. The red commit alone yields ‘ABCDEFG’. Applying only the blue commit on top of the red commit yields ‘ABEFCDG’, it has moved the contents ‘EF’ in between B and C. Note that this is different than a deletion and an insertion because the independent green commit gets moved along with it to yield ‘ABEHFCDG’.

The green and blue commits do not conflict, since a path (the cogent path) exists that traverses all of both of their edges. However our definition of the verge doesn’t work here since there are cycles. There is no way to advance the verge past B, because to advance to C we’d need to have already advanced it past F, which is obviously impossible.

To remedy this we need to allow the verge to ignore pairs of edges from the same commit that are going in opposite directions. In this case it means that the verge can advance past C because the incoming blue edge can be ignored.



Here is how the verge advances through several nodes in this graph.

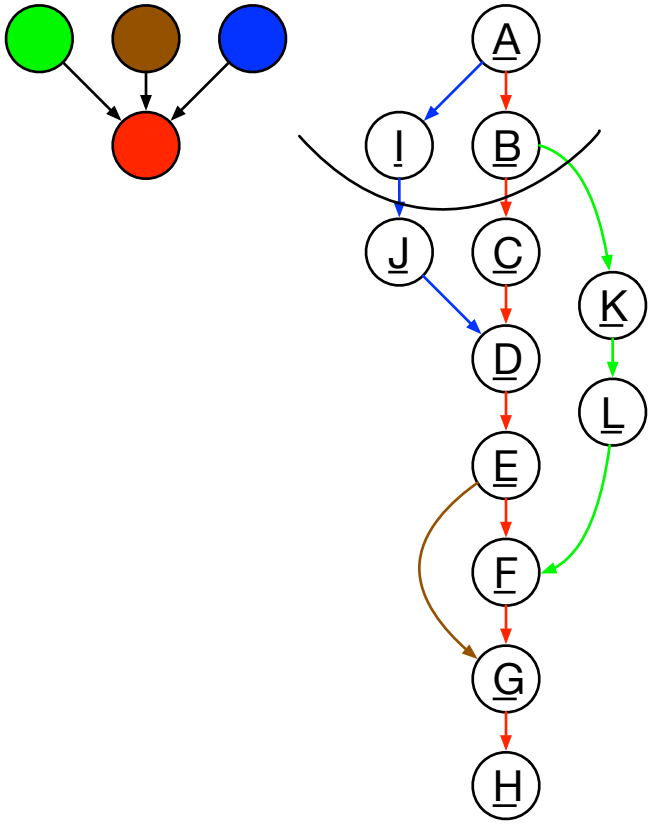
1. The verge is cutting one red and three blue edges, but we can ignore the reverse blue edge and one of the forward blue ones. Since the verge is only cutting blue and red edges there is no conflict.

2. The verge is cutting one red, one green, and two blue edges. We can ignore both blue edges though since one is a reverse edge, so effectively it is only cutting red and green edges, so there is no conflict.

3. This is similar to 2, except the verge is cutting a different green edge this time.

4. A very ordinary cutting of a red and blue edge.

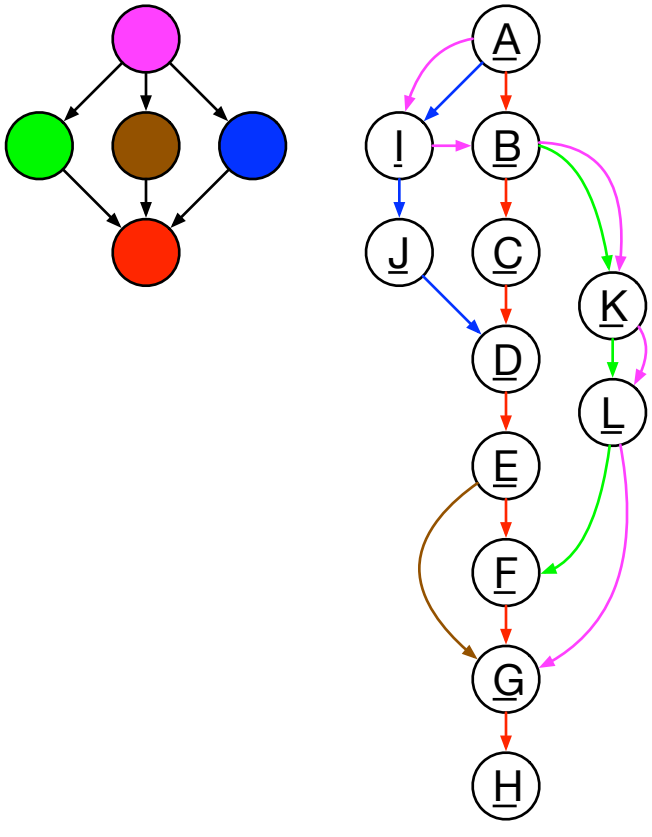
Note that from 2 we could not have advanced the verge past F since it requires that the verge passes H first.



If there are no conflicts then there is little work to do, the cogent path is easy to follow. However, if there is a conflict we need to indicate to the user what is conflicted. The range of the conflict should begin and end with nodes that are not in conflict.

In this case the verge has found two edges (blue and green) that conflict. To find the range of the conflict start with the conflicted edges and trace backward, always taking the most dependent edges, until they converge. This is the beginning of the conflict. Similarly, trace forward until they converge and this is the end of the conflict. Here when we trace backward we end up at A. When we trace forward we end up at G, at which point we don't even have incoming edges from the green or blue commits. The brown commit also conflicts with the green one, and so the range of the commit includes all three of them.

To display this to users we must display the full range of the conflict (from A to G) for each commit so the user can decide how to resolve them. Since the blue and brown commits do not conflict and, except for the green commit, would have merged together without trouble, we can actually show those together. So in this case we could show the user 'AIJDEG' and 'ABKLFG'.



Here is one possible way of resolving the conflict in this graph. The magenta commit has incorporated part of the blue commit, all of the green commit, and also deleted the F, which was what the brown commit did. Now the cogent path clearly exists since there is a magenta path all the way from A to G, which was the range of our conflict.