A graph contains no odd cycle $\iff$ the graph is bipartite.

No odd cycle $\implies$ bipartite

A human-proof would start something like this: Create a bipartition A, B as follows:

- Choose an arbitrary vertex v_0. Put v_0 in A.
- If a vertex has odd distance from v_0, put it in B.
- If a vertex has even distance from v_0, put it in A.

…but there's an issue. How can a computer think of the non-obvious step of sorting vertices by distances from an arbitrary "central" vertex? It seems that as a preliminary step, we use a "forced-construction" tactic. (This is what both I and another person solving it did, and also seems likely for a computer algorithm to realize is useful).

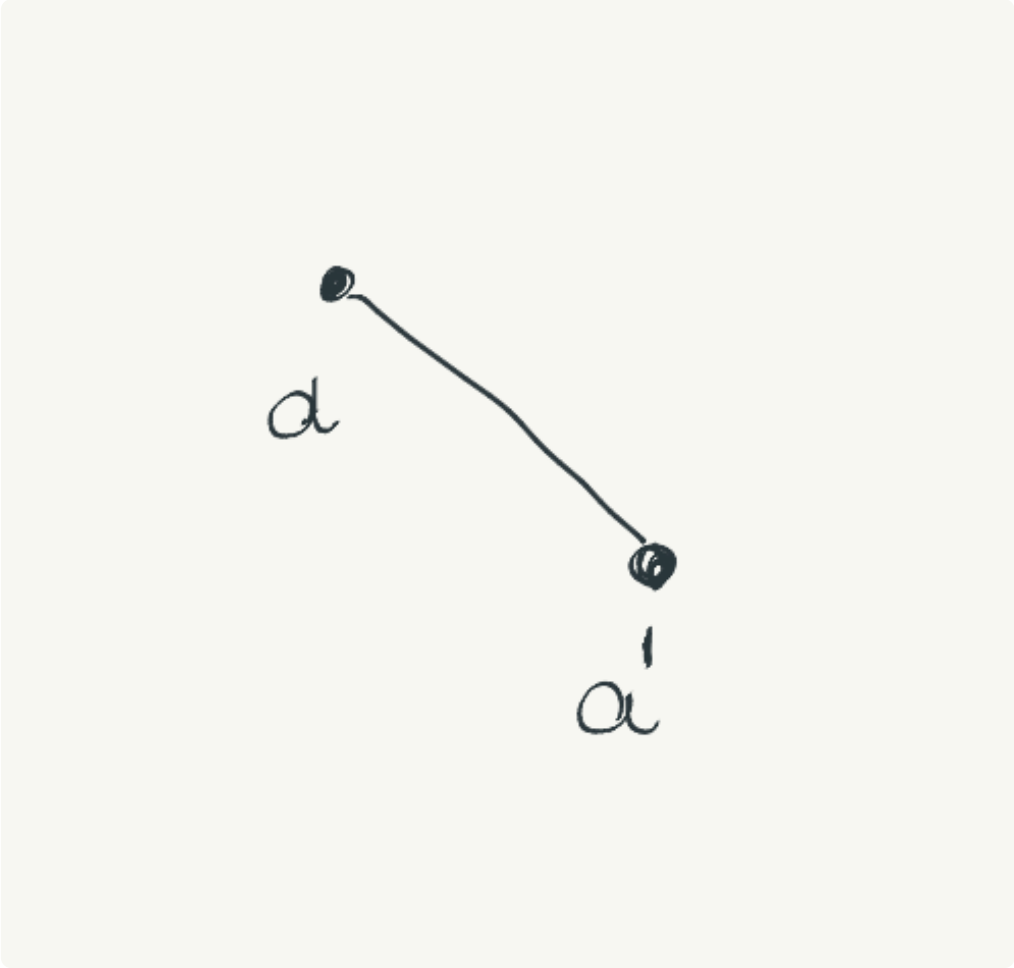
That is, a computer-proof might start something like this:. Let us force-construct a bipartition A, B:

- Choose an arbitrary vertex v_0. Put v_0 in A.
- Let all the neighbours of v_0 (call that set of neighbours N_1) be in B.
- Let all the neighbours of elements in of N_1 that haven't yet been assigned a partition be in A (call the set of these vertices N_2).
- …continue this process. It must partition all vertices because the graph is connected. It must terminate because there are finitely many vertices.

Now, is there a need to turn this inductive construction into a closed-form one? It turns out that there is, as exemplified in this next step.

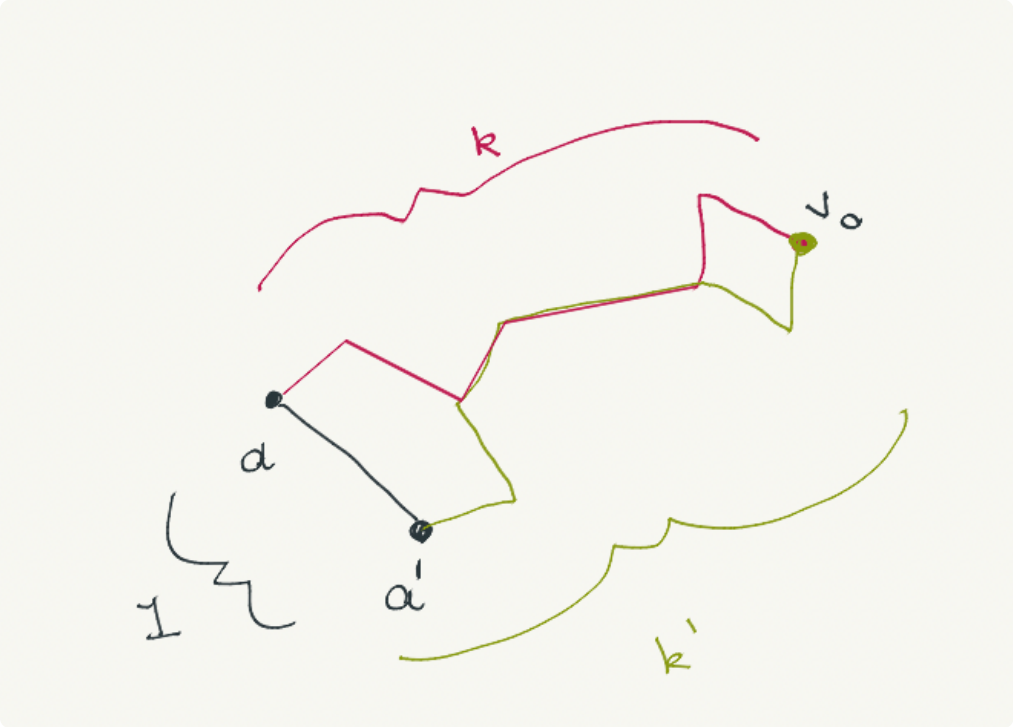
We have to prove that this is a valid bipartition. We want to prove that for every two vertices in A, there exists no edge between them. Again, we have to prove a "there does not exist" statement, so this is a good enough reason to switch to proving by contradiction.

- Suppose for the purposes of contradiction that there are two vertices a and a' in A with an edge between them.



- The algorithm would notice it has to create a metavariable representing the odd cycle. To populate this metavariable with vertices, it would try to look for the paths or cycles in the graph where it has some information about the length.
- Given the human construction, you can see that there's
  - a path of even length from a to v_0

- a path of even length from a' to v_0
- a path of length 1 from a to a'



- And we should be able to combine these paths to create an odd cycle.

But in order for the computer prover to see this, it should turn its inductive construction into a closed form. Since it's not the focus of this document, let's assume we have some induction-to-closed-form module working. How would the computer know that turning it's construction to a closed-form one would help prove this lemma about cycle lengths?

Here's where conflict-guided reasoning comes in . The drive to find a closed-form (for both of the humans who did this problem) was sparked when coming up with examples of constructions that satisfy the inductive definition. We might want it to work similarly for the computer. In particular, the computer **might notice over and over again that the paths it comes up with from the central vertex to that vertex a takes on even values, then adds that as a lemma, and proves that lemma using the closed-form construction module.**

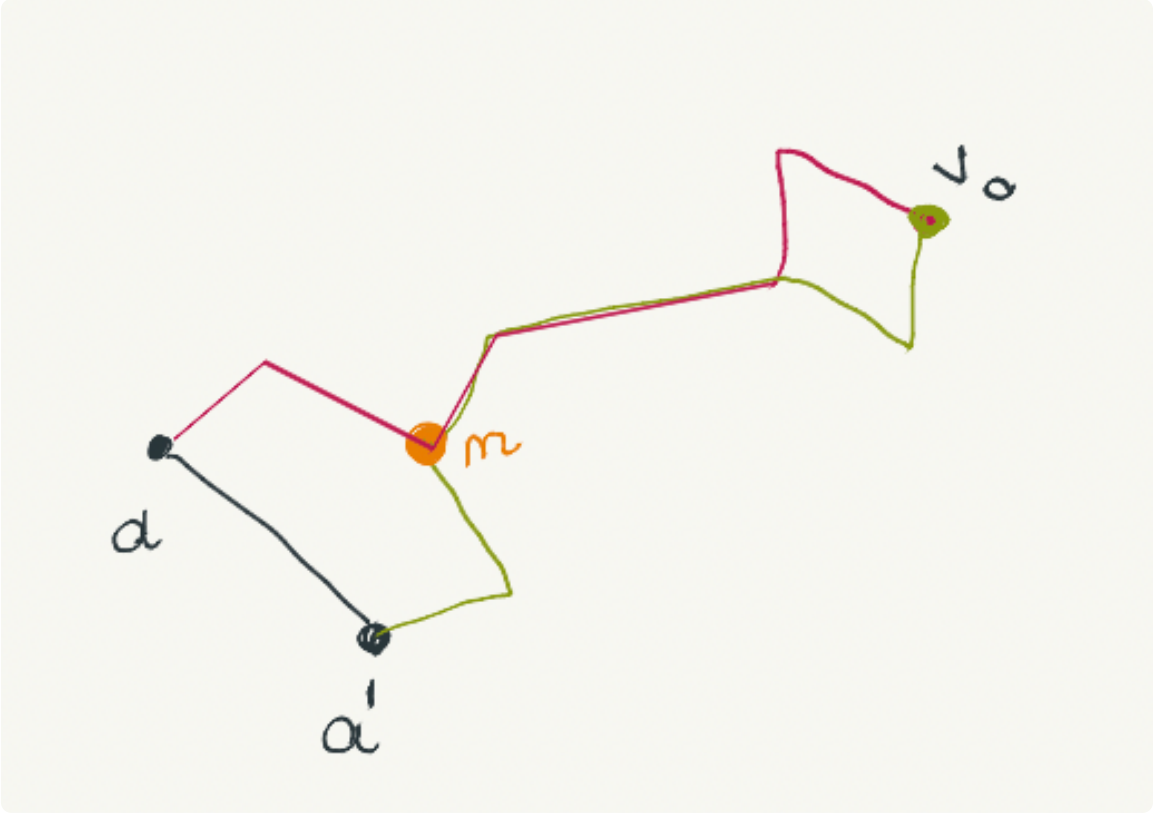
So now our computer system has access to three paths in the

hypothesis:

- a path of even length k from a to v_0
- a path of even length k' from a' to v_0
- a path of length 1 from a to a'

We now want to prove that we get an odd cycle. And if concatenating the paths from a to v_0, then v_0 to a', then a' to a contains an odd cycle, then we are done.

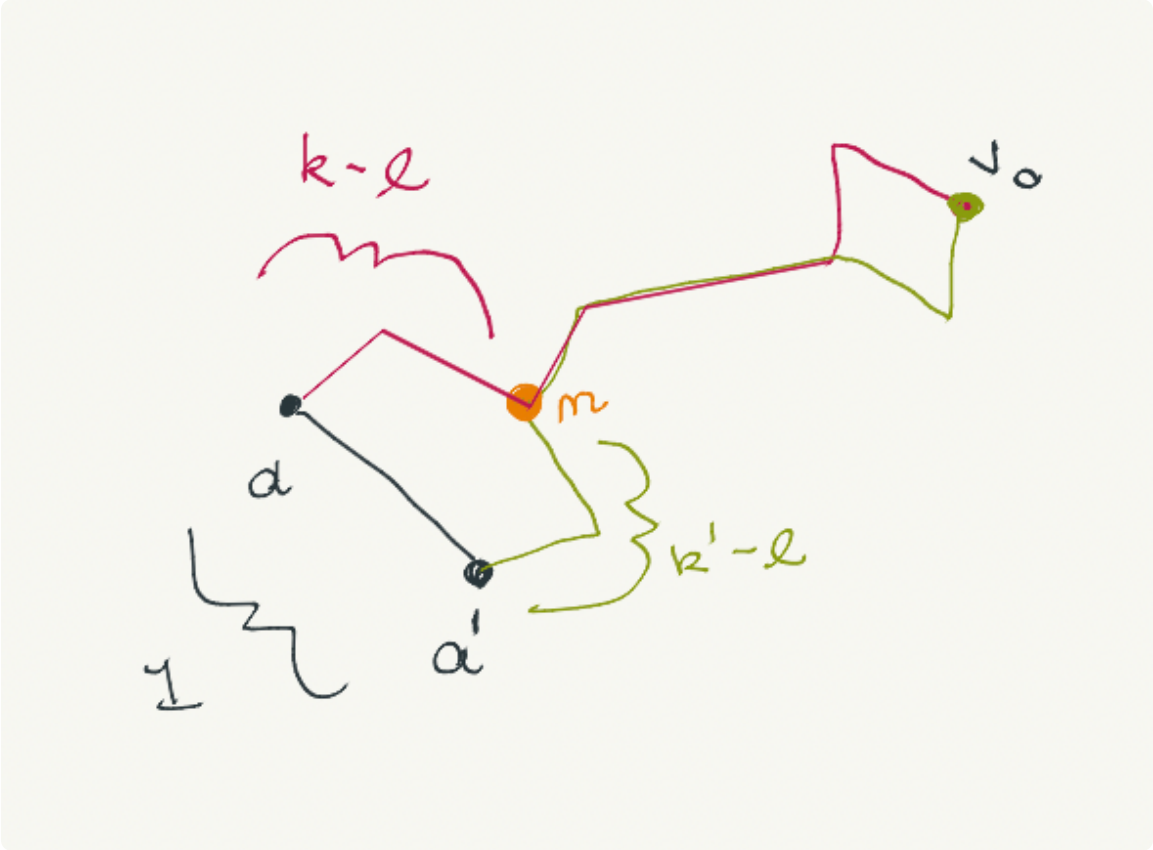
But **here, conflict-guided reasoning plays a role again**. Because the computer will try to prove it, and when it fails to, might try to generate counterexamples — is it really true that concatenating those three paths gives you a cycle? And it will quickly find out, hopefully, that it is not — those three paths can intersect. But, hopefully, it will notice that the first point of intersection m between the a -to-v_0-path and the a' -to -v_0-path (m is, that is, the point of intersection closest to a) is the one that is consistently involved in an odd cycle in numerous examples. Then **it can try to prove a lemma: there is always an odd cycle involving that vertex** m.



Indeed, it will find that by letting the distance from v_0 to m be ℓ, then the cycle formed by concatenating the a-to-m -path with the m-to-a'-path and then a'-to-a- path, we have a cycle of length

$$k + k' - 2\ell + 1,$$

which is odd, a contradiction.



The prover can then notice the same argument works if we consider an edge between two vertices $b, b' \in B$.

Bipartite $\implies$ no odd cycle

This is the simpler direction, one that does not require counterexample-lemma oscillation, but I'm just including it for completeness.

1. We have to prove a "does not exist" statement, which is a good enough reason to prove by contrapositive. So instead we want to prove: *there exists an odd cycle* $\implies$ *not bipartite.*

2. Consider any vertex of the odd cycle of length $2k + 1$. Call it v_0. Try to force-construct a partition by putting v_0 in A.
    - Going around the cycle in one direction, v_0's neighbouring vertex must be in partition A.
    - But now two vertices in A neighbour each other. So the forced construction doesn't work. So no construction can work.

This "pick_arbitrary _first _vertex" and then "force _construct" tends to come up quite a bit in graph theory. So, it might be interesting to eventually think of how to turn a "greedy algorithm" proof into a more explicitly inductive proof, and then how to turn an inductive proof into a closed form proof (in this case, a proof that says vertices an even distance away from v_0 are in A, and odd distance away are in B.)