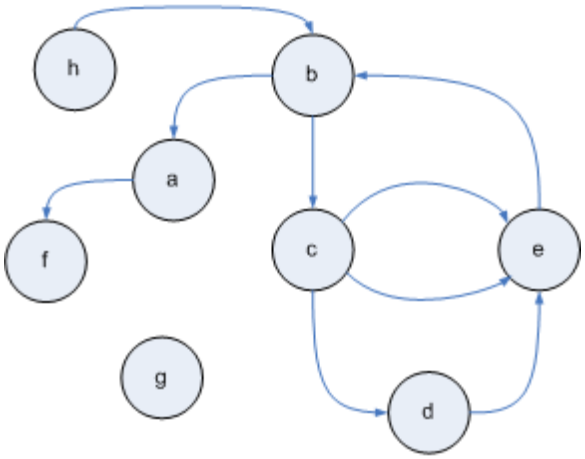
This page last changed on Jan 10, 2007 by dswood.

Graph - Hop Count

Session Objectives	Session Overview
By the end of this session students should ...	**Activity** • Lab - Hop Count • Lab - Min Hop Count

Lab - Hop Count



Ask students to come up with the hop count, for example from 'b' to 'e'.

Things to look for

- This is not min hop count, just any hop count.
- Throwing exception versus returning -1. It makes sense to throw an exception from the public method, but not for flow control. In our flow control being unable to reach a destination down one particular search path is not an exceptional condition. More this will cause the entire search to end early and you will only be able to find your first neighbor. Since we are in a private method and the implementation is hidden, it is alright to use a magic number, just don't expose it to the world.
- Must check terminal conditions in *recursive question*, not the *original question*
- Lots of redundancy: for example calls to canReach() or multiple calls to hopsTo(destination, seenNodes) with the same parameters instead of remembering the first result.
- Name your methods with your return value. It makes it more readable and thus basically self documenting. For example instead of *countHopsTo()* use *hopsTo()*.

Ask the students if there is any further duplication in the code. They should point out that canReach() and hopsTo() are very similar. Ask students how they could get rid of it. Students may suggest separating the tree traversal algorithm from what action is taken on each node (ie. Visitor). This is a lot of work. If you are seeing duplication and can't figure out how to remove it, the first step is to make the code look as similar as possible. Give the students time to do this.

Ask for suggestions again. Direct students towards something very simple. The best answer is to have canReach() simply call hopsTo().

Lab - Min Hop Count

Ask students to come up with the min hop count.

Things to look for

- Will need to clone seenNodes so that you can find the best path through a single node. In other words searching one path through that node will not later prevent you from searching a potentially shorter path through that node.
- Solutions that pass the current min hop count around. This is a valid solution but there is a trade off. Keeping track of minHops in the for loop is more concise code, versus the efficiency of stopping paths early.
- When extracting methods in Eclipse the return value is not always generalized correctly and may be HashSet<> when it should be Set<>.
- Optimization: since add() returns a boolean indicating whether something really was added, it can be combined with the check to see if seenNodes already contains the node. The tradeoff may be readability.