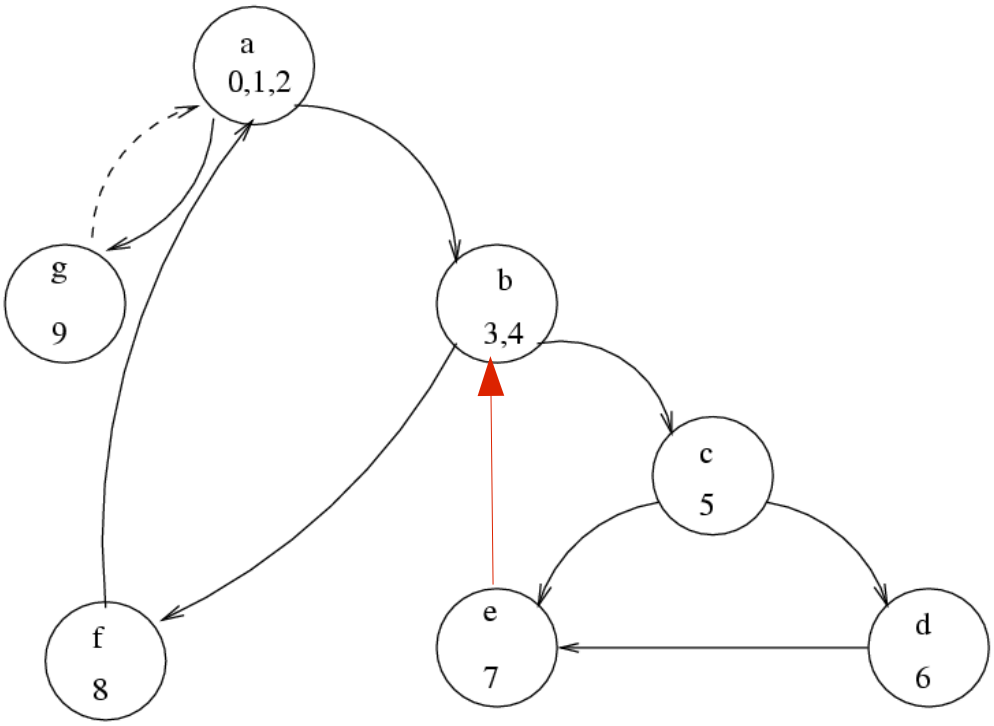
Clarification: cyclomatic complexity

First: consider the following example of cyclomatic complexity, from Jalote Ch. 7 -- as mentioned in class, I think there's a piece missing in the textbook example, which was added in class and is also included here.

I'm keeping Jalote's example bubble-sort code fragment as given in the text, in spite of its icky formatting, since he uses line numbers based on that icky formatting:

```
0. {
1.      i=1;
2.      while (i<=n) {
3.         j=i;
4.         while(j <= i) {
5.              If (A[i]<A[j])
6.                  Swap(A[i], A[j]);
7.              j=j+1; }
8.      i = i+1; }
9. }
```

And, here is the control flow graph, with the missing edge shown in red:



Discussing/explaining this control flow graph a bit:

- the lines of code from the beginning of the fragment to the first decision, the outer-while loop condition, numbered 0, 1, and 2, correspond to the node marked with **a** and the line numbers 0, 1, 2
- the lines of code from the beginning of the body of the outer-while to the next decision, the inner-while loop condition, numbered 3 and 4, correspond to the node marked with **b** and the line numbers 3, 4
- the line of code with the next decision, the if-statement condition, numbered 5, corresponds to the node marked with **c** and the line number 5.
- the line of code with the if-statement action (and implied "end of if-statement body"), numbered 6, corresponds to the node marked with **d** and the line number 6.
- the line of code that increments j (and implied "end of inner-while-loop body"), numbered 7, corresponds to the node marked with **e** and the line number 7.
- the line of code that increments i (and implied "end of outer-while loop body"), numbered 8, corresponds to the node marked with **f** and the line number 8.
- the line of code that ends this fragment -- just the closing brace! -- numbered 9, corresponds to the node marked with **g** and the line number 9. This is where control is transferred when the outer while loop is completed.

Hopefully, that's so far, so good.

Now, consider the edges in this graph:

- at the first decision, the outer-while loop condition, control can pass to either the body of the outer while loop (if the condition is true) or after the loop (if the condition is false). So, there are two edges from node a/lines 0, 1, 2:
  - one from node a/0,1,2 to node b/3,4 (the beginning of the outer while-loop body)
  - and one from node a/0,1,2 to node g/9 (after the outer-while body)
- at the second decision, the inner-while loop condition, control can pass to either the body of the inner while loop (if the condition is true) or after the inner loop body (if the condition is false). So, there are two edges from node b/lines 3, 4:
  - one from node b/3,4 to node c/5 (the beginning of the inner while-loop body)
  - and one from node b/3,4 to node f/8 (after the inner-while body)
- at the third decision, the if-statement within the inner while loop, control can pass to either the if-action (if the condition is true) or after the if-action (if the condition is false) -- note that there is no else part. So, there are two edges from node c/line 5:
  - one from node c/5 to node d/6 (the if-action)
  - and one from node c/5 to node e/7 (the statement after the if-action)
- node d/6 only has one edge from it, because it is the end of the if-action, and control passes to node e/7 (the statement after the if-action)
- node e/7 has no edge leading out of it! That's the omission I was talking about -- here, there SHOULD be an edge from node e/7 to node b/3,4, since control passes from the end of the inner loop to the beginning of the inner loop. I've included this edge in red.

- node f/8 only has one edge from it, to node a/0,1,2, because it is the end of the outer loop, and control passes to the beginning of the outer loop.
- and, node g/8 has an implied/dotted edge back to node a/0,1,2, I'm guessing because once the fragment is done, you perhaps could start it over if you'd like...? p. 218 implies this is to result in a strongly-connected graph (adding an edge from the exit node to the entry node).

I hope the above makes it easier to understand the discussion on pp. 218-219 of Jalote. You should read over that again, and note:

- the cyclomatic complexity of a module is defined to be the cyclomatic number of such a control-flow graph.
- the cyclomatic number *V(G)* of a control-flow graph *G* with *n* nodes, *e* edges, and *p* connected components is:

  $V(G) = e - n + p$

  - in the given graph, there are 10 edges, 7 nodes, and 1 connected component (ah -- because starting at a, you can reach every other node and come back to node a?) [notice that you only get 7 edges if you add the edge I've shown in red -- that's one reason I think this is an omission.]

    V(G) = 10 - 7 + 1 = 4
- It turns out that the cyclomatic complexity of a module (or cyclomatic number of its control flow graph) is equal to the maximum number of linearly-independent circuits in the graph, where a circuit is linearly independent if no circuit is totally contained in another circuit or is a combination of other circuits.
  - There are indeed 4 such circuits in this graph: (and again notice how several of these include that red edge):

    linearly-independent circuit 1: b c e b

    linearly-independent circuit 2: b c d e b

    linearly-independent circuit 3: a b f a

    linearly-independent circuit 4: a g a
- Here's the money-shot: it can also be shown that the cyclomatic complexity of a module is the number of decisions in the module plus one, where a decision is effectively any conditional statement in the module!
  - so -- after all that! -- you can ALSO compute the cyclomatic complexity simply by counting the number of decisions in the module and adding 1 --
  - here, we have 3 such decisions, for the outer while, inner while, and if, plus 1 again gives us 4!
- Can you see how this cyclomatic number can be one quantitative measure of a module's **complexity**? (Notice I said a quantitative measure of **complexity**, not of size!) Two modules might have the same number of lines, but quite different cyclomatic numbers, and it seems reasonable to argue that the module with more decisions is more complex than one that is simply a sequence of statements.
  - McCabe, who proposed this measure, also proposed that the cyclomatic complexity of modules should, in general, be kept below 10.
- Experiments indicate that the cyclomatic complexity is highly correlated to the size of the module in LOC; interestingly, it has also been found to be correlated to the number of faults found in modules.