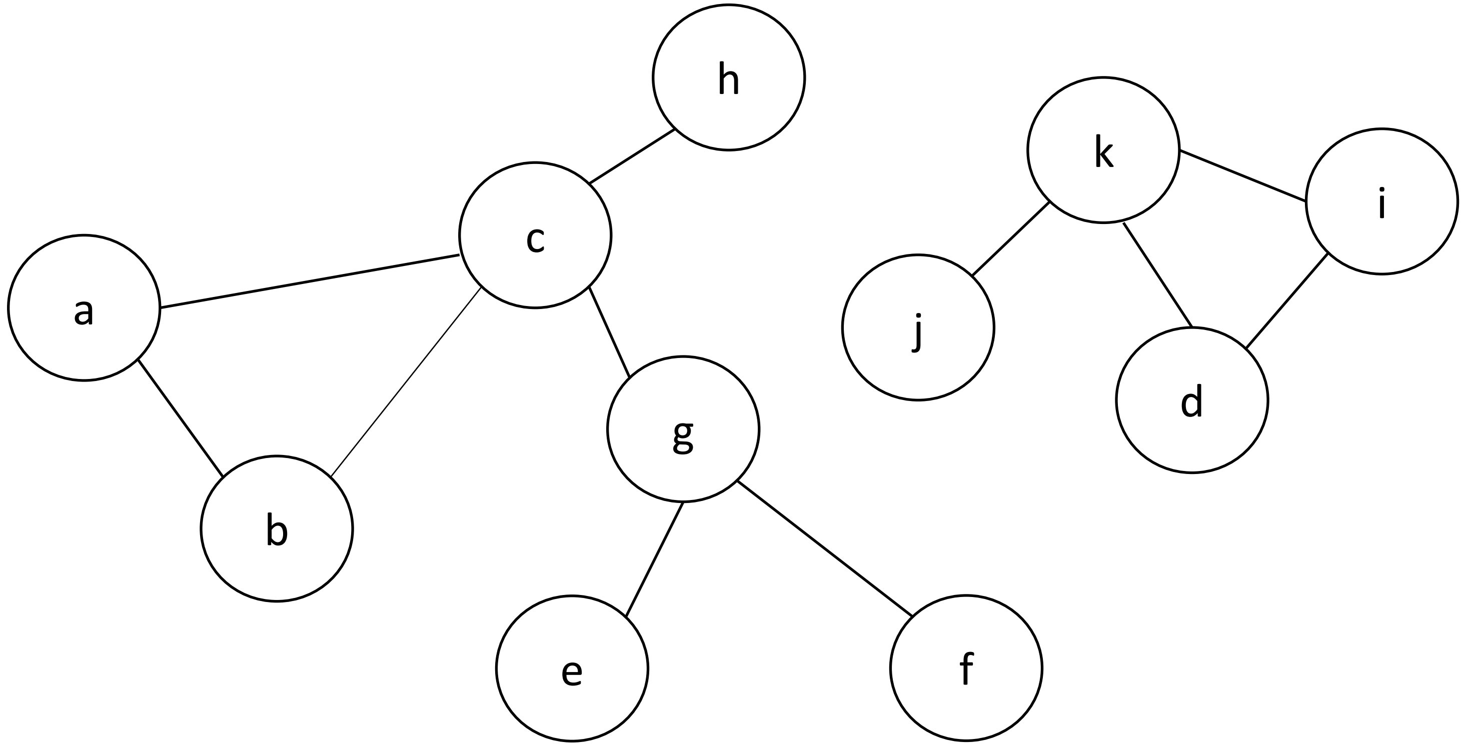
Graph A



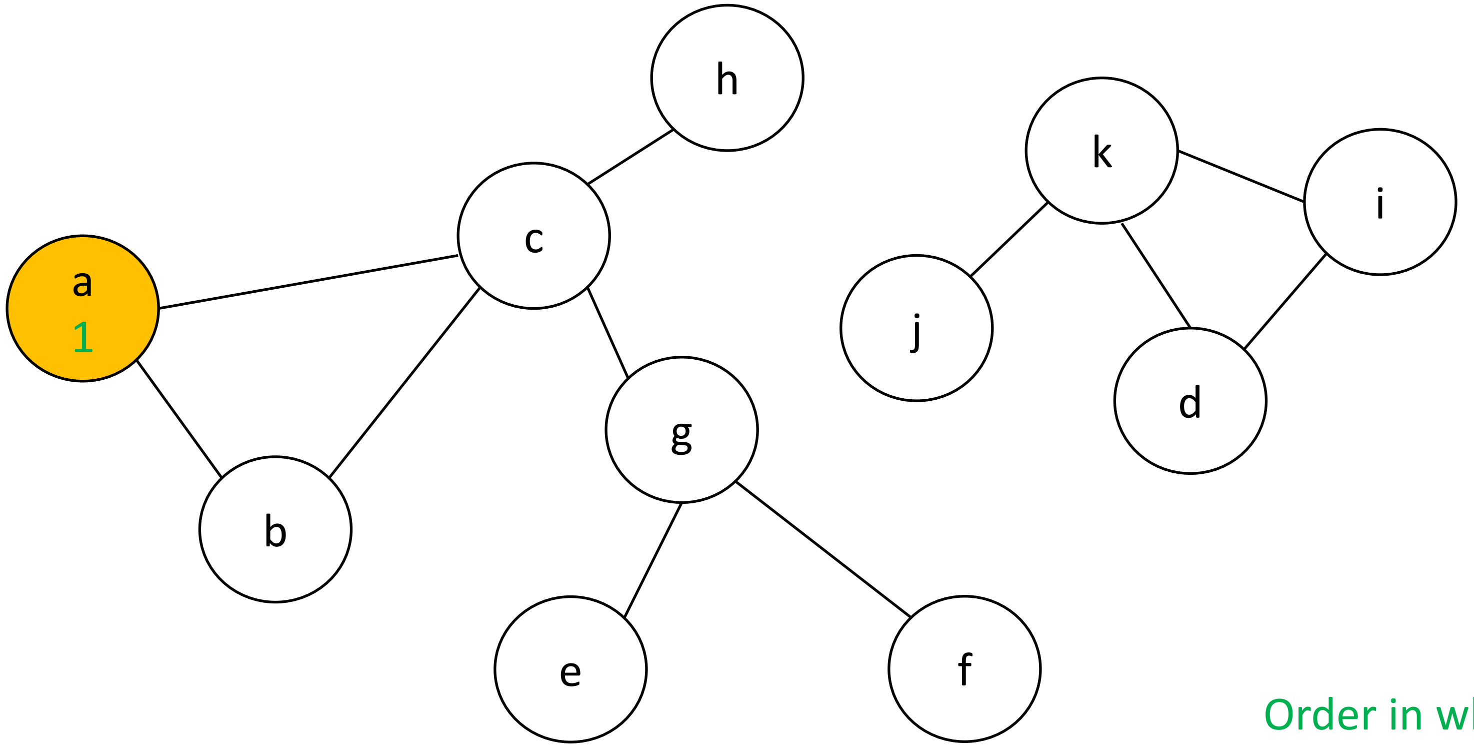
In executing DFS on A, we'll start from vertex *'a'.*

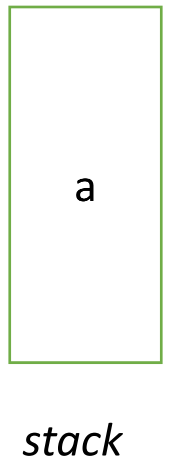
Also, we will use alphabetical ordering for tie resolutions

Can you try it yourself first before looking at mine?

LET'S BEGIN DFS

Step 1. create a stack, mark ‘a’ as visited and add it to the top of the stack

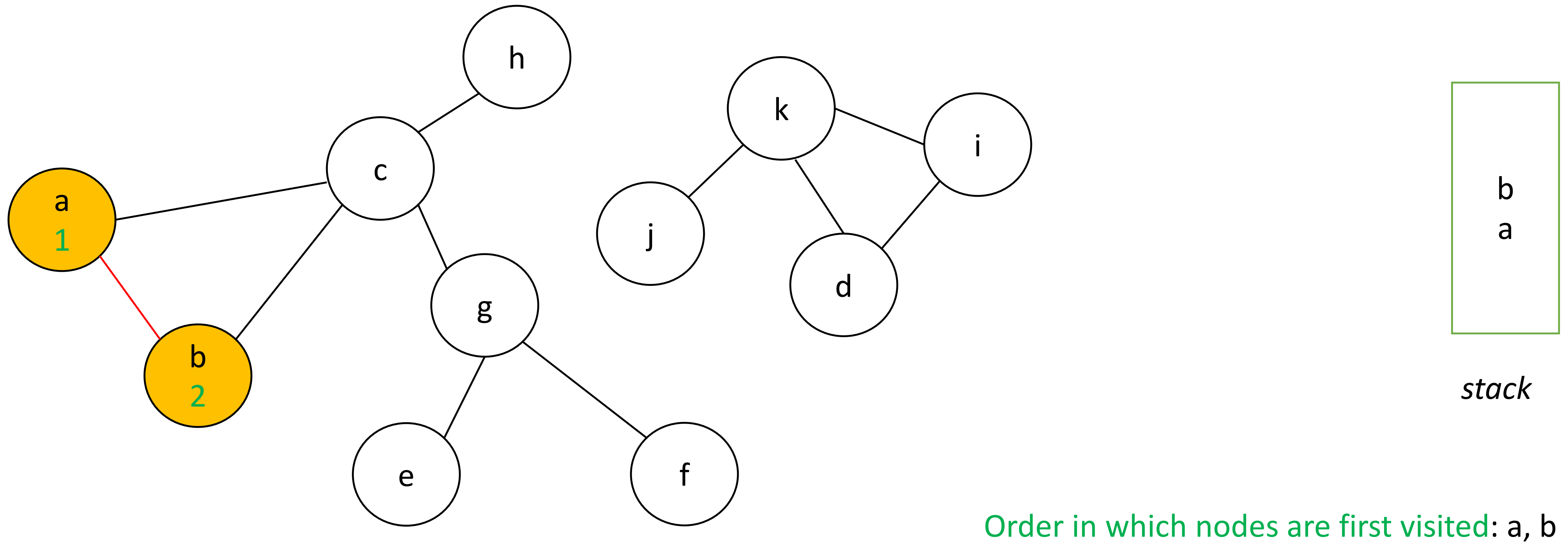




Order in which nodes are first visited: a,

Order in which nodes become dead ends:

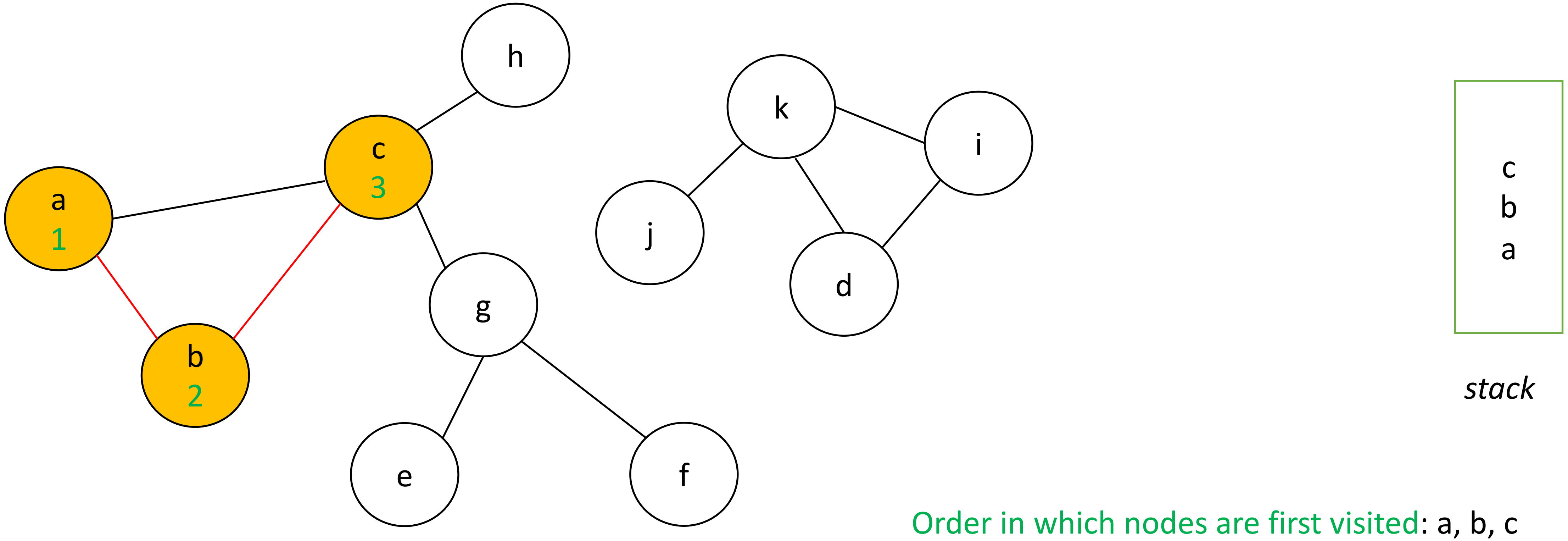
Step 2. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is 'a') which is not filled yellow. Since there are two of such vertices, we invoke our tie resolution. This is gives us 'b'. Mark 'b' and add it to the top of the stack



Order in which nodes are first visited: a, b

Order in which nodes become dead ends:

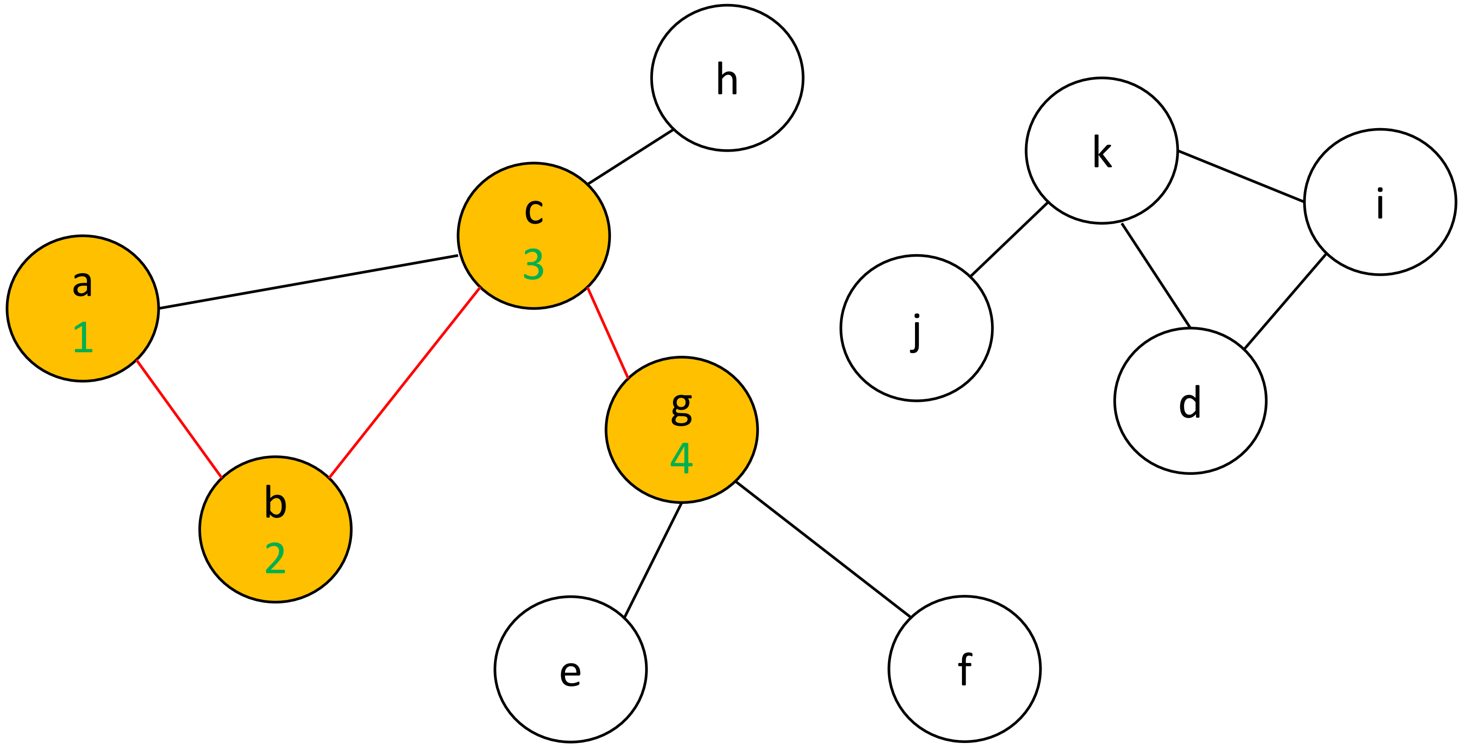
Step 3. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is currently ‘b’) which is not filled yellow. This is gives us ‘c’ as our next vertex. Mark ’c’ and add it to the top of the stack

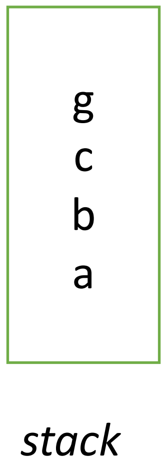


Order in which nodes are first visited: a, b, c

Order in which nodes become dead ends:

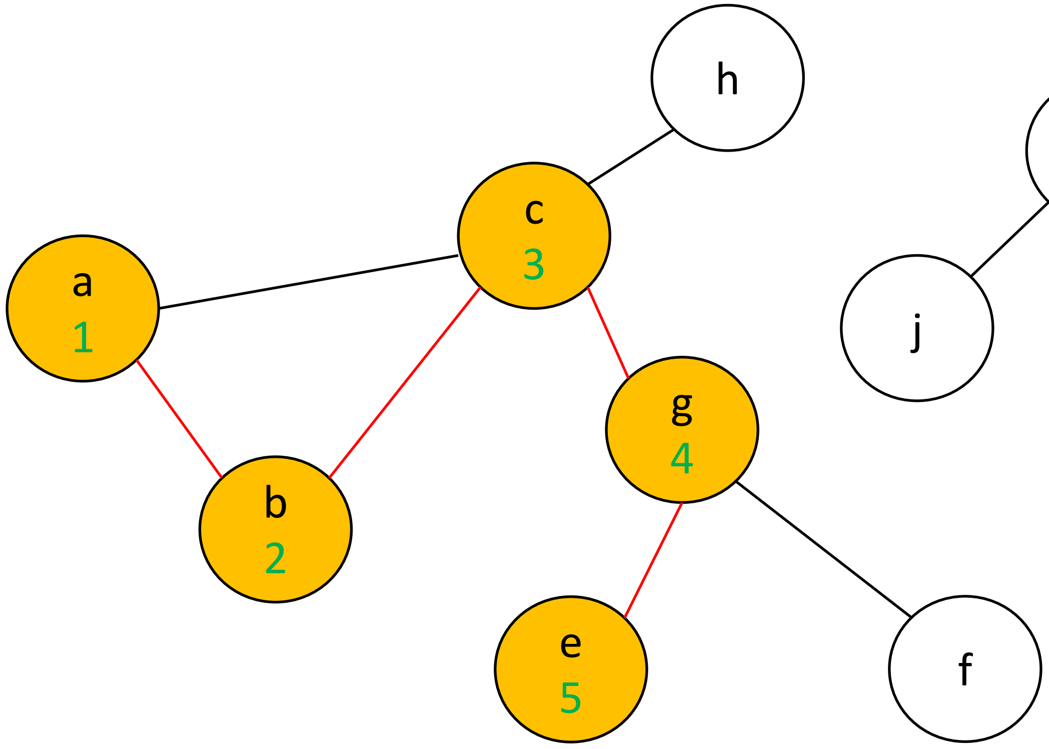
Step 4. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘c’) which is not filled yellow. This gives us ‘g’ as our next vertex. Mark ‘g’ and add it to the top of the stack

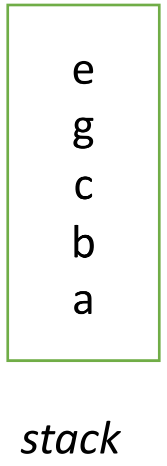




Order in which nodes are first visited: a, b, c, g
Order in which nodes are become dead ends:

Step 5. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘g’) which is not filled yellow. This gives us ‘e’ as our next vertex. Mark ‘e’ and it to the top of the stack

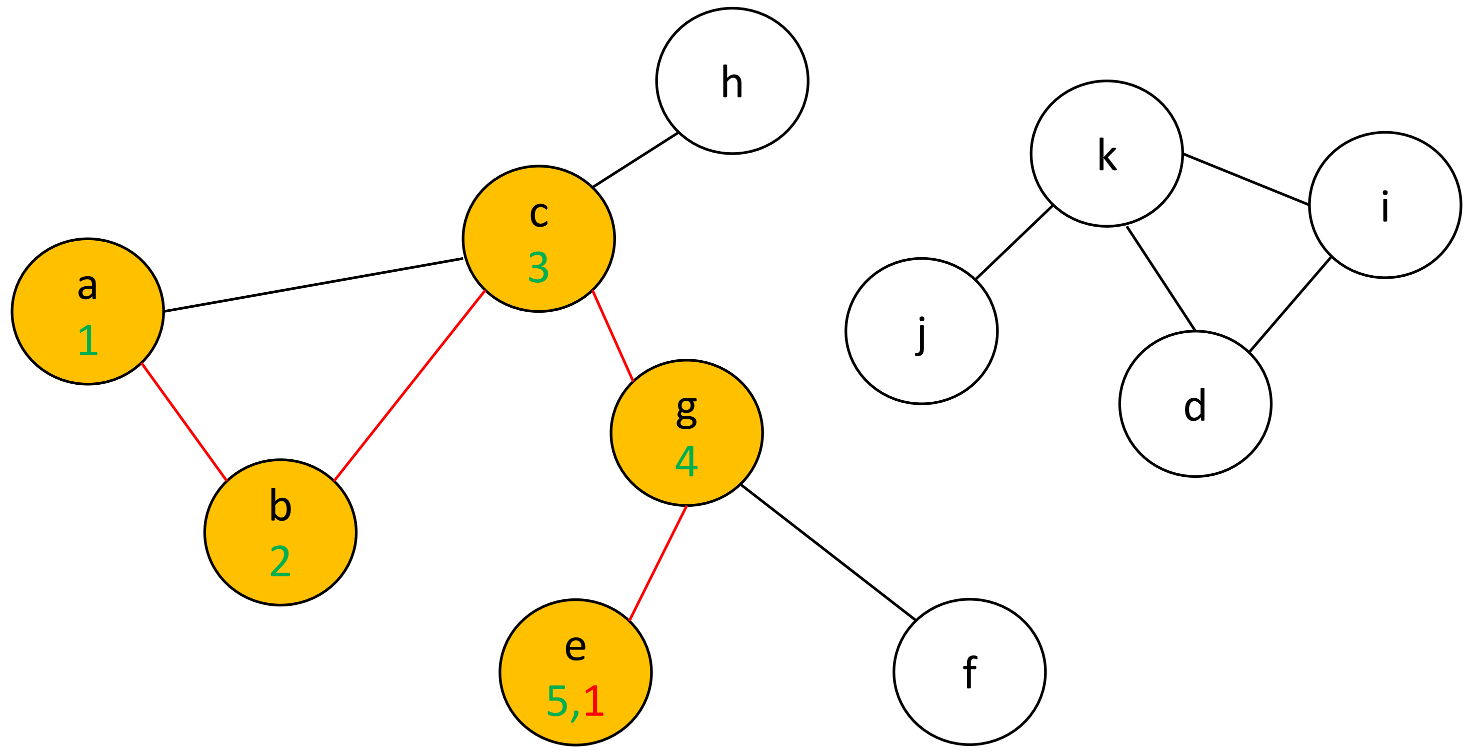


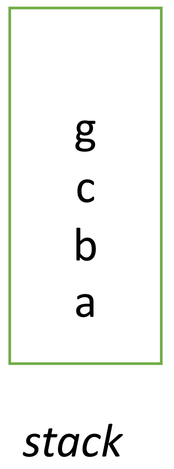


Order in which nodes are first visited: a, b, c, g, e

Order in which nodes are become dead ends:

Step 6. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is currently ‘e’) which is not filled yellow. There is no such vertex. We’ve encountered our first dead-end. Pop ‘e’ off the stack

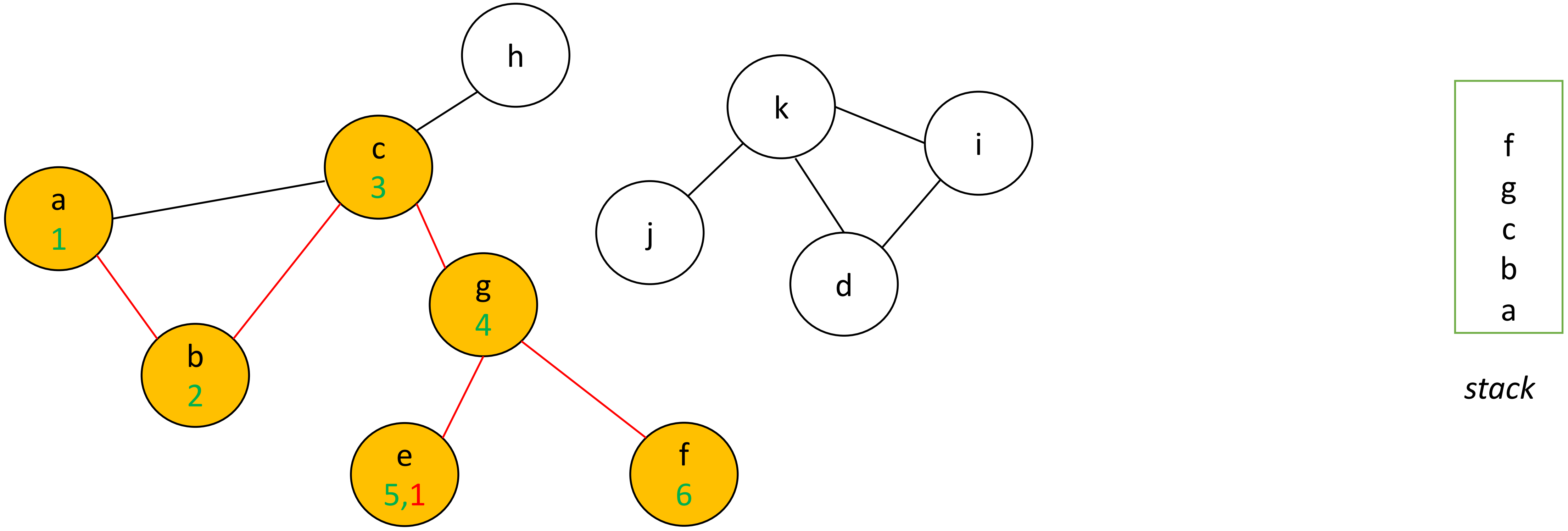




Order in which nodes are first visited: a, b, c, g, e

Order in which nodes are become dead ends: e

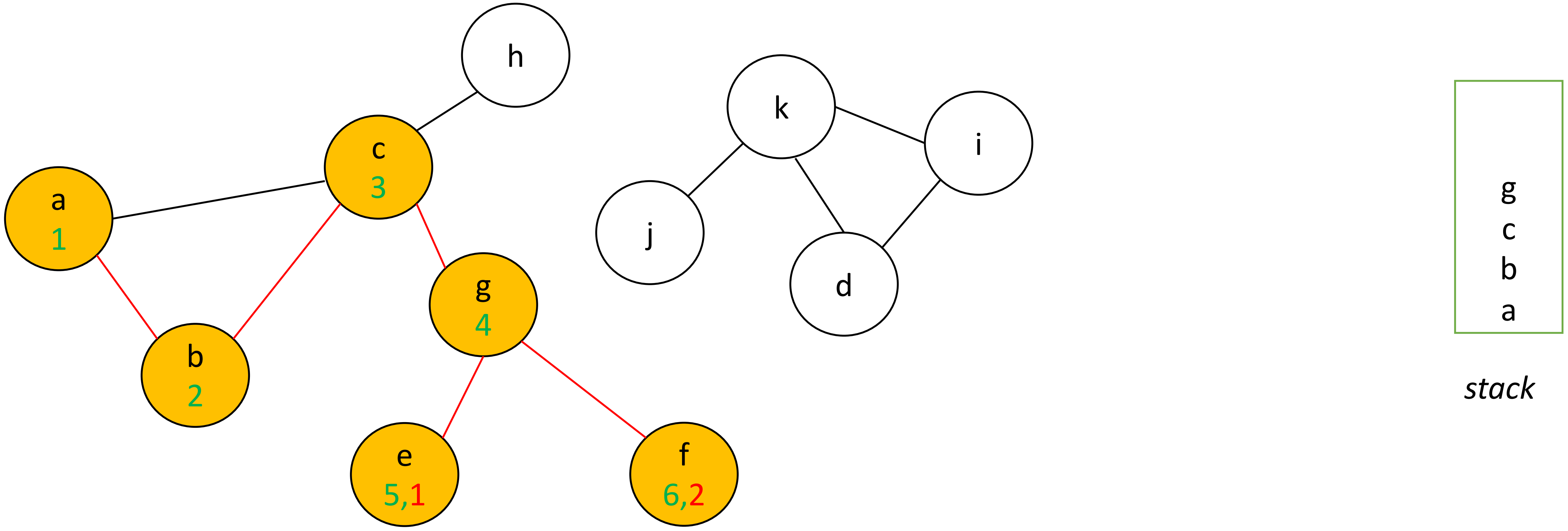
Step 7. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘g’) which is not filled yellow. This gives us ‘f’. Mark ‘f’ and it to the top of the stack



Order in which nodes are first visited: a, b, c, g, e, f

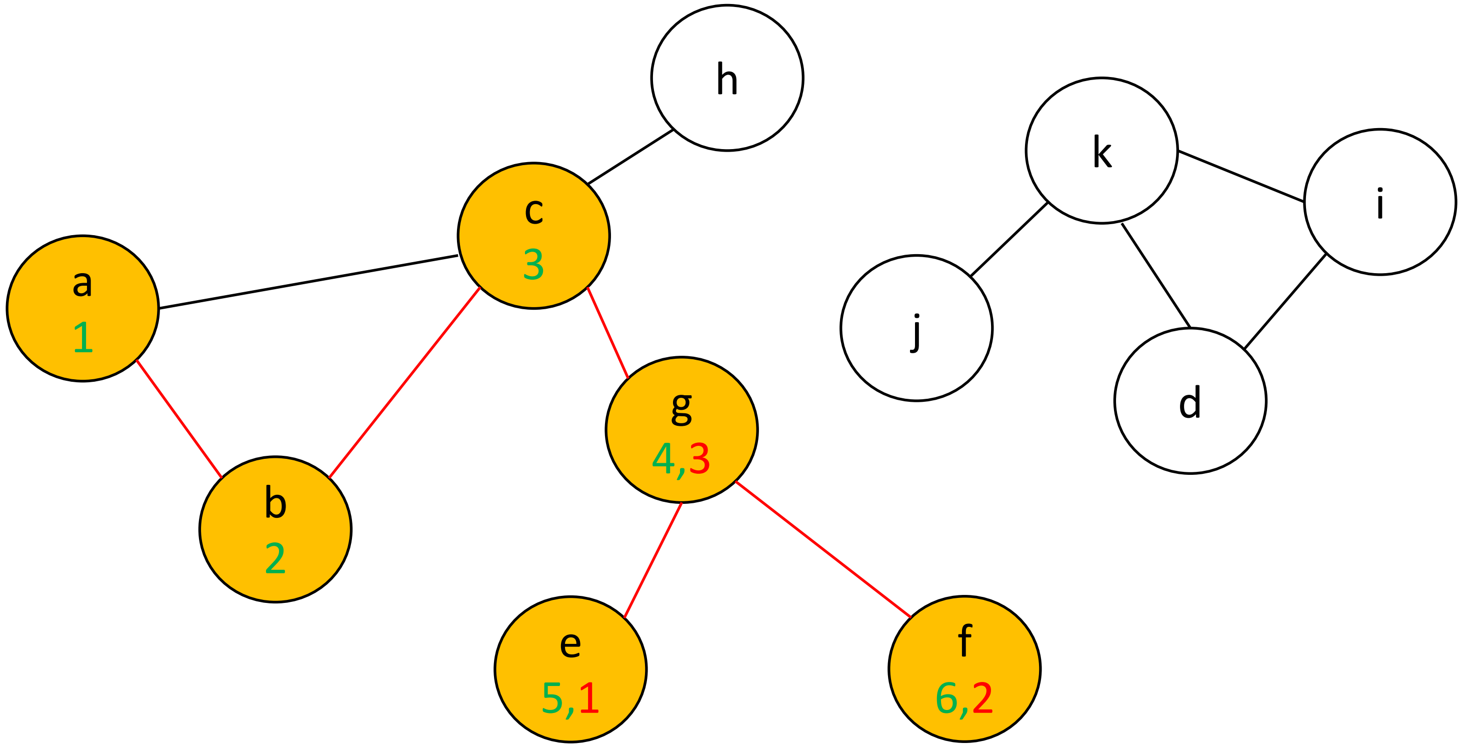
Order in which nodes are become dead ends: e

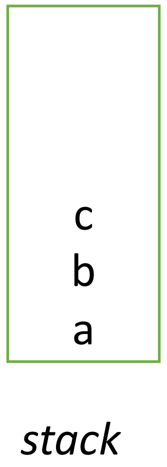
Step 8. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘f’) which is not filled yellow. But there’s no such vertex. We have our second dead-end. Pop ’f’ off the stack



Order in which nodes are first visited: a, b, c, g, e, f
Order in which nodes are become dead ends: e, f

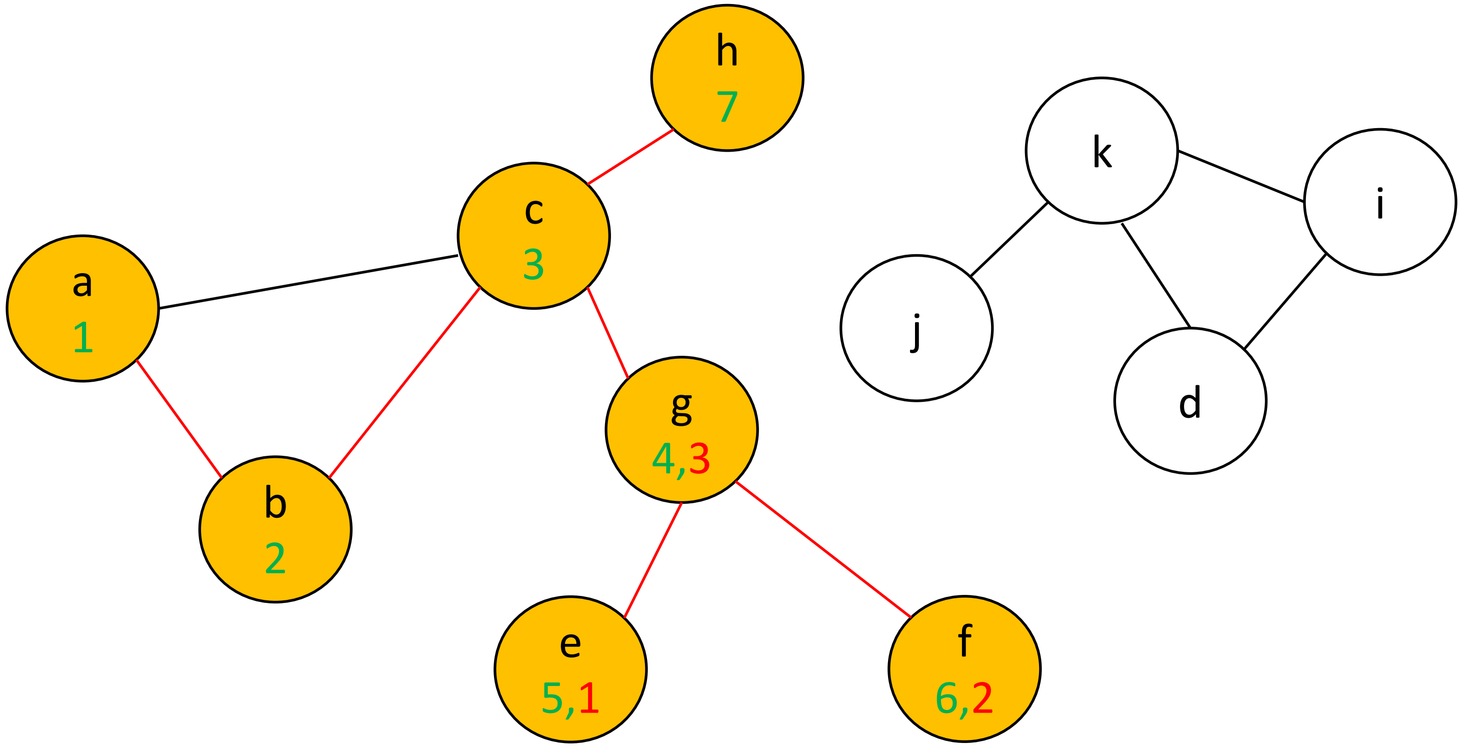
Step 9. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘g’) which is not filled yellow. But there’s no such vertex. We have our 3rd dead-end. Pop ’g’ off the stack

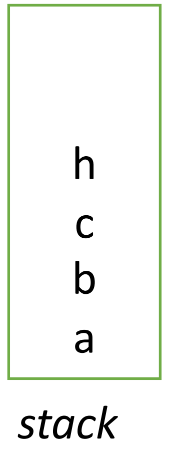




Order in which nodes are first visited: a, b, c, g, e, f
Order in which nodes are become dead ends: e, f, g

Step 10. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘c’) which is not filled yellow. This gives us ‘h’. Mark ‘h’ and add it to the top of the stack.

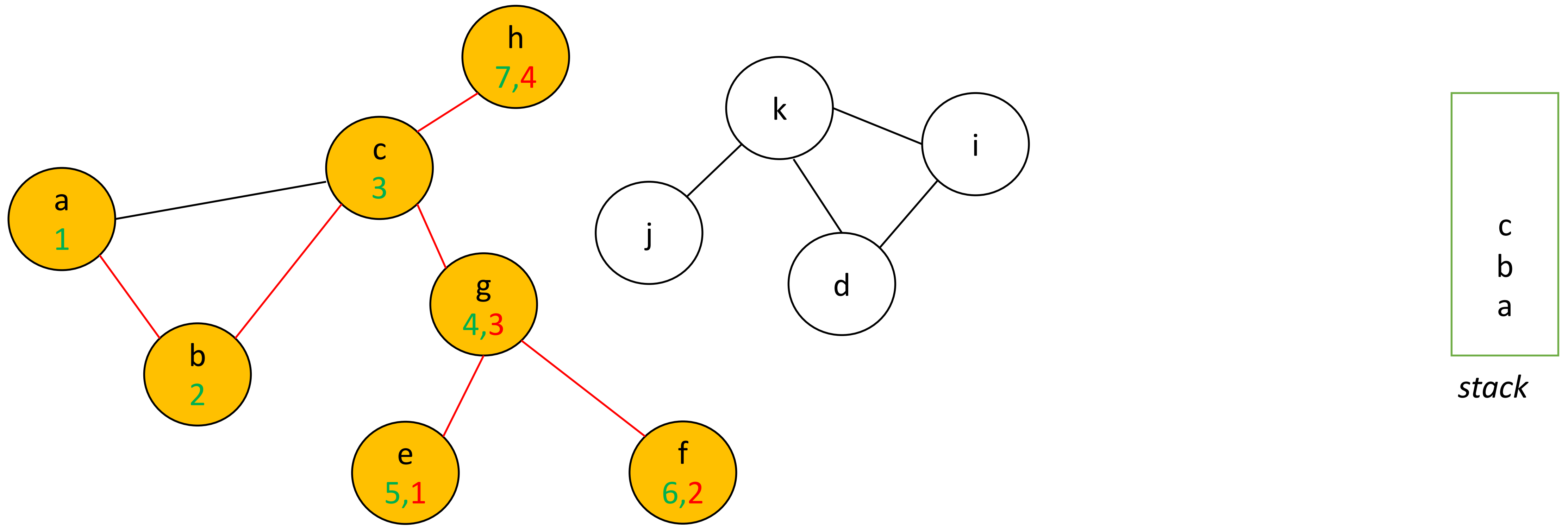




Order in which nodes are first visited: a, b, c, g, e, f, h

Order in which nodes are become dead ends: e, f, g

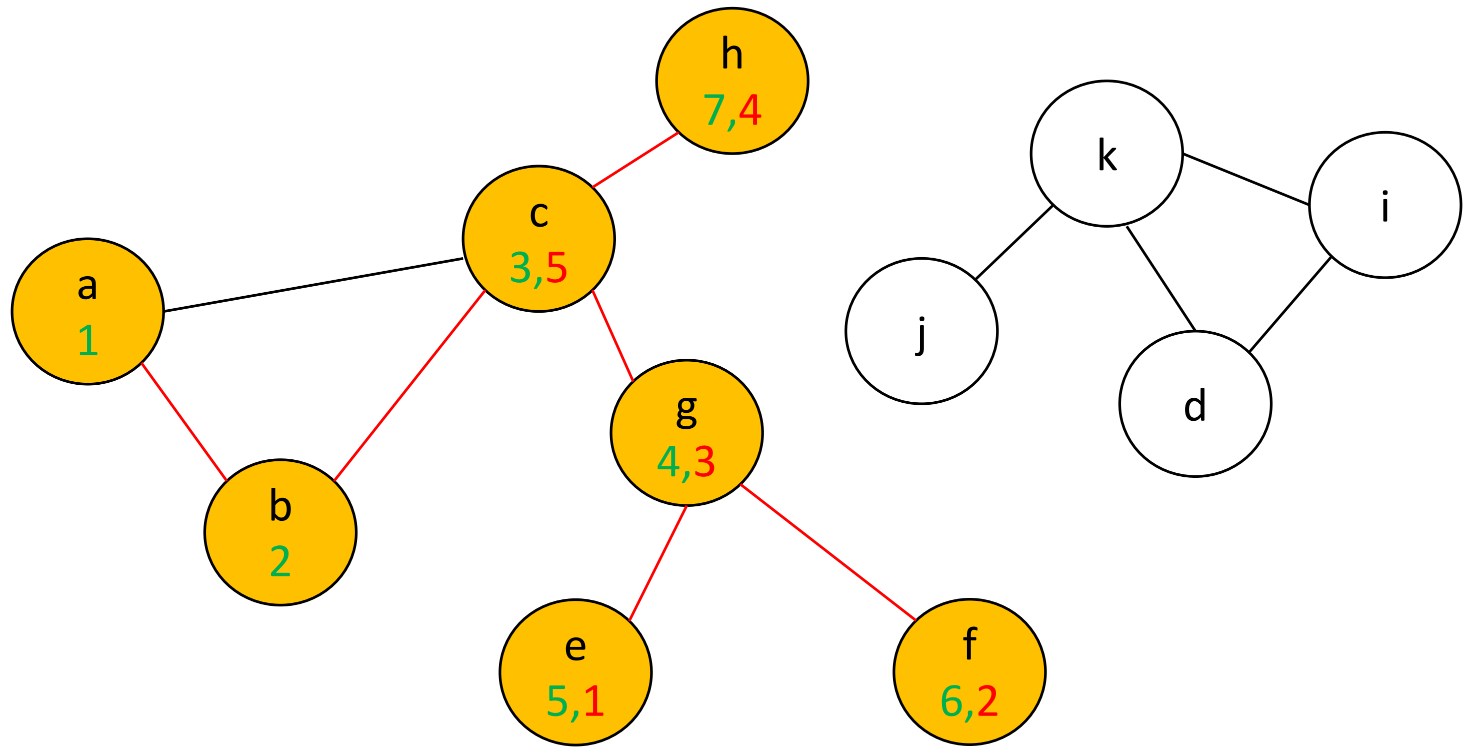
Step 11. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘h’) which is not filled yellow. But there’s no such vertex. Pop ‘h’ off the stack

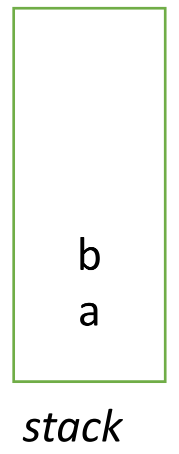


Order in which nodes are first visited: a, b, c, g, e, f, h

Order in which nodes are become dead ends: e, f, g, h

Step 12. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘c’) which is not filled yellow. But there’s no such vertex. Pop ‘c’ off the stack

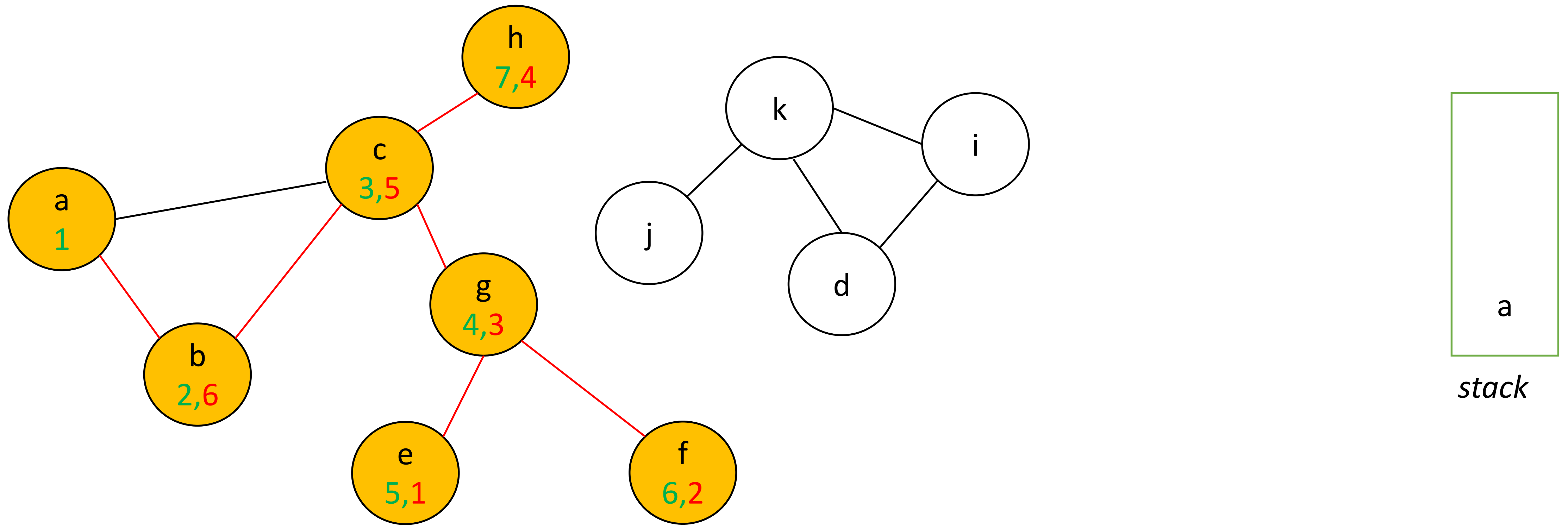




Order in which nodes are first visited: a, b, c, g, e, f, h

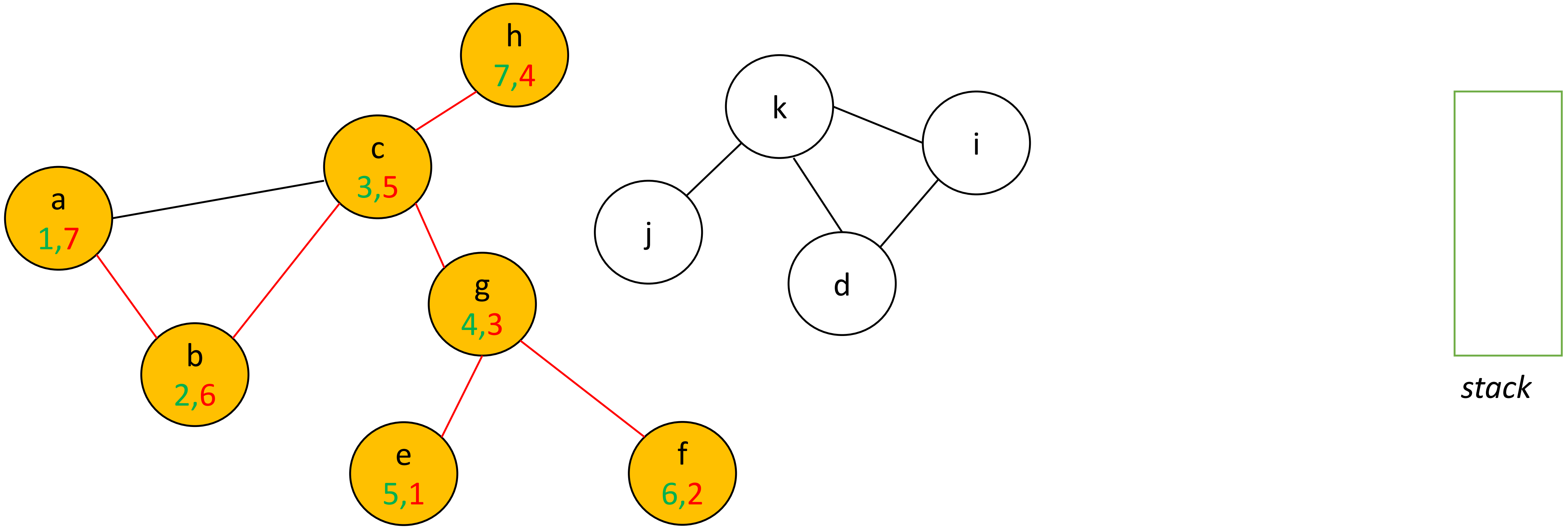
Order in which nodes are become dead ends: e, f, g, h, c

Step 13. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘b’) which is not filled yellow. But there’s no such vertex. Pop ‘b’ off the stack



Order in which nodes are first visited: a, b, c, g, e, f, h
Order in which nodes are become dead ends: e, f, g, h, c, b

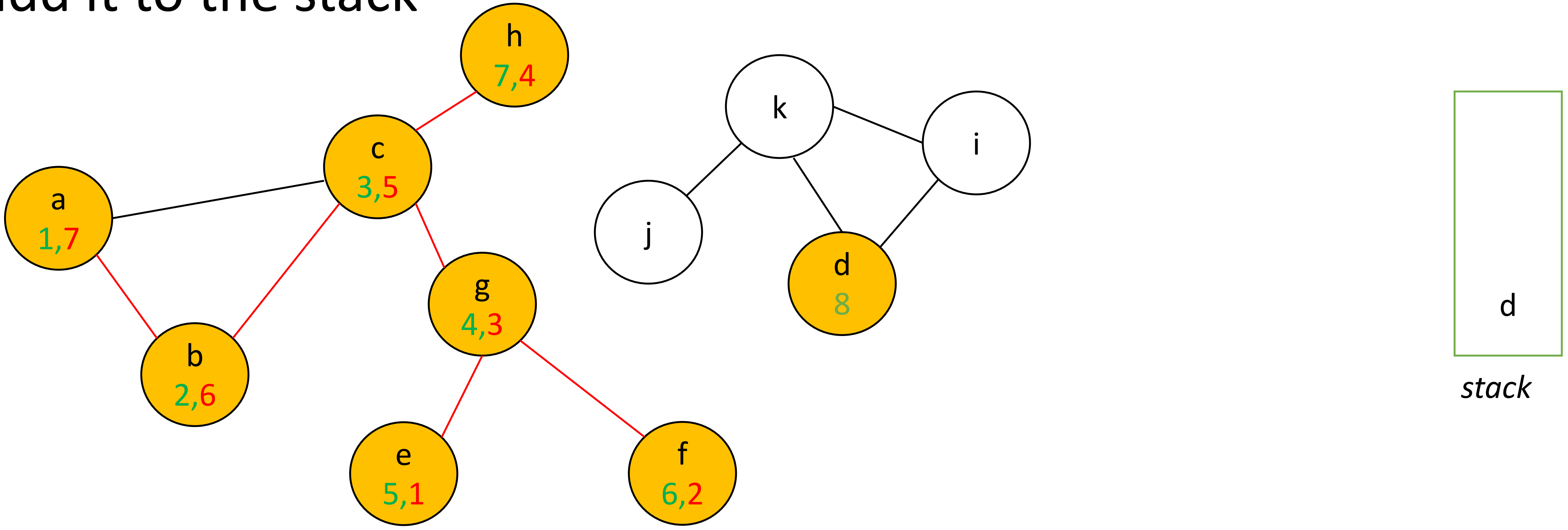
Step 14. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘a’) which is not filled yellow. But there’s no such vertex. Pop ‘a’ off the stack



Order in which nodes are first visited: a, b, c, g, e, f, h

Order in which nodes are become dead ends: e, f, g, h, c, a

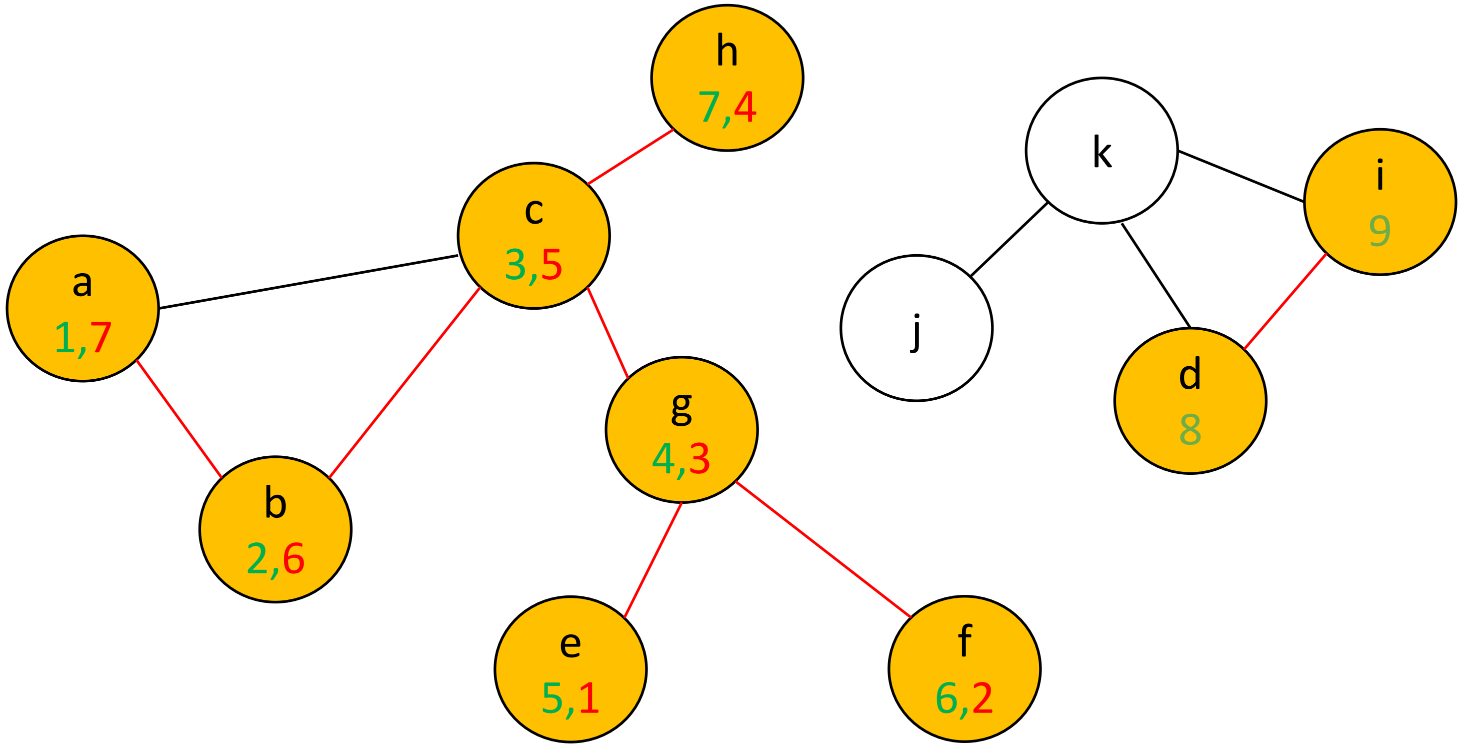
Step 15. Since our stack is empty, it means we have finished traversing one connected component of the graph. But there could be other connected components. To verify this, we check whether all the vertices are filled? The answer is no. So we start from one of the unfilled vertices. Using our tie resolution strategy, this gives us ‘d’. Mark ’d’ and add it to the stack

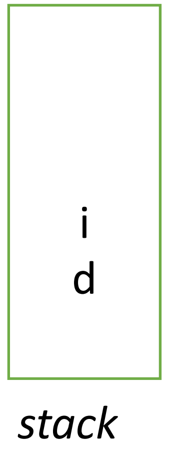


Order in which nodes are first visited: a, b, c, g, e, f, h, d

Order in which nodes are become dead ends: e, f, g, h, c, a

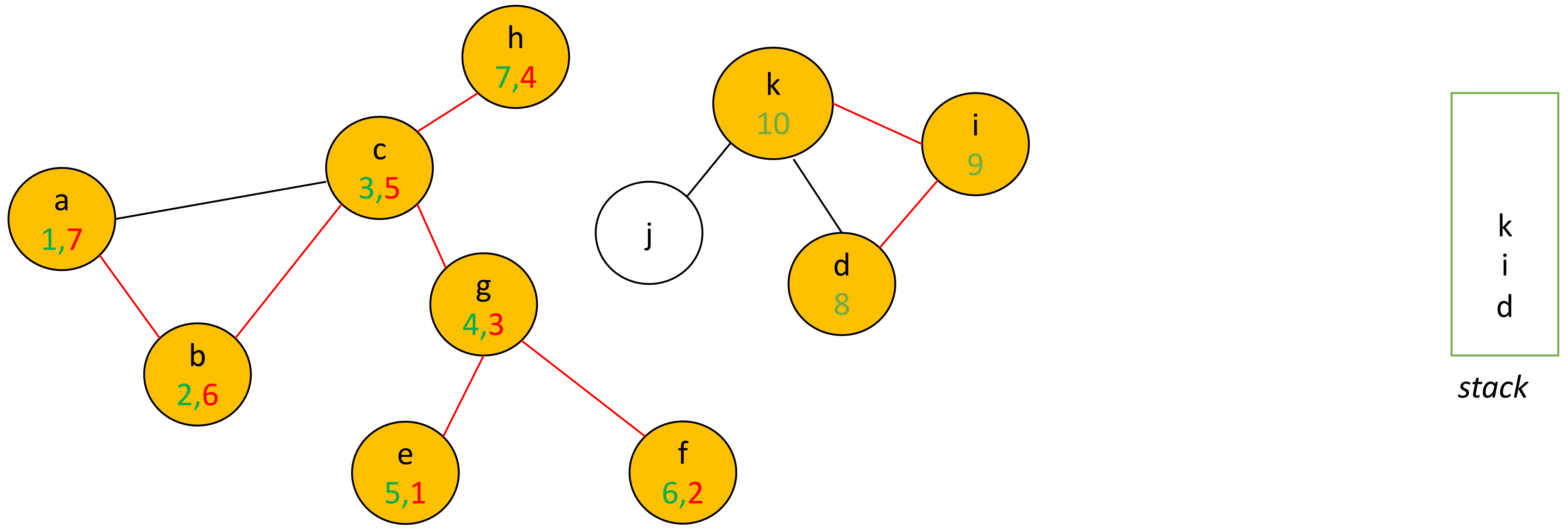
Step 16. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘d’) which is not filled yellow. Using our tie resolution strategy, this gives us ‘i’. Mark ‘i’ and add it to the stack





Order in which nodes are first visited: a, b, c, g, e, f, h, i
Order in which nodes are become dead ends: e, f, g, h, c, a

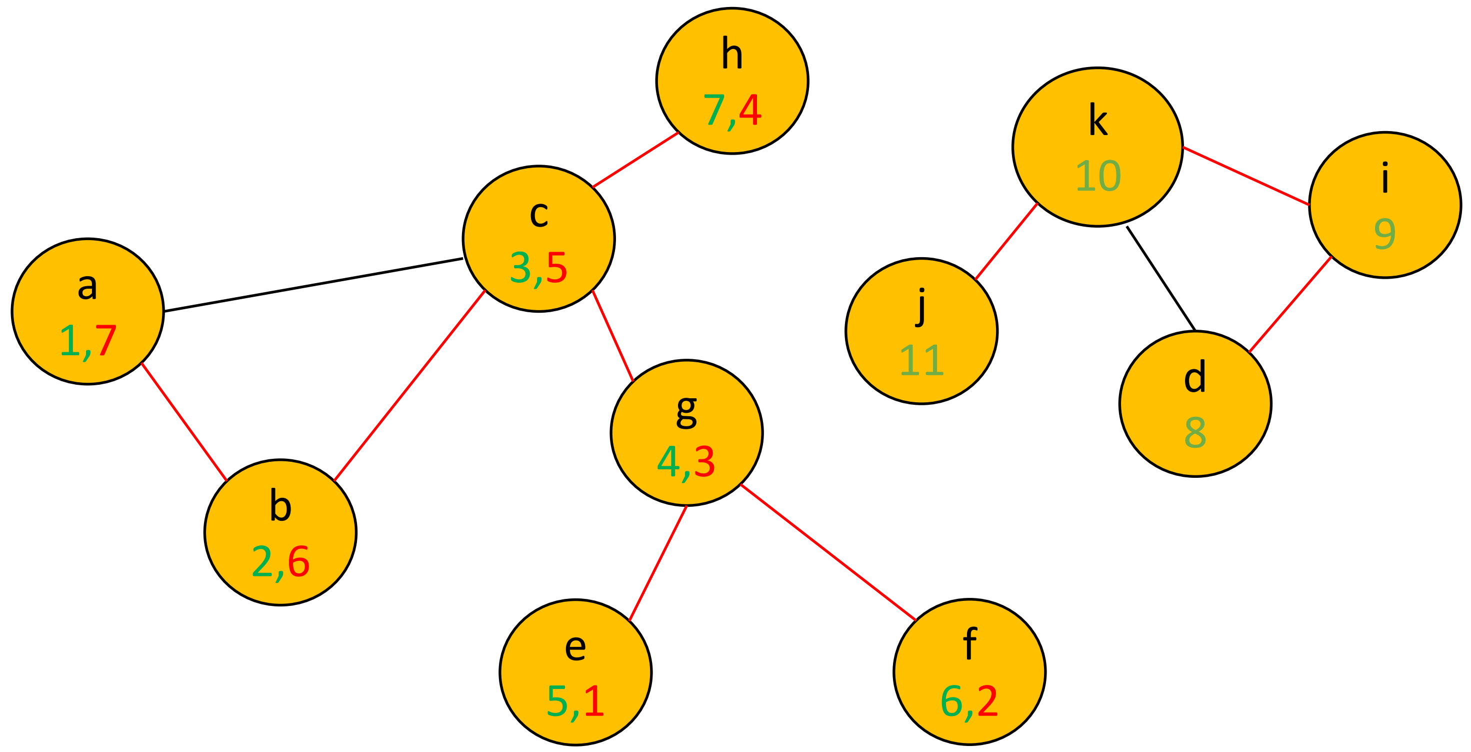
Step 17. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘i’) which is not filled yellow. This gives us ‘k’. Mark ‘k’ and add it to the top of the stack

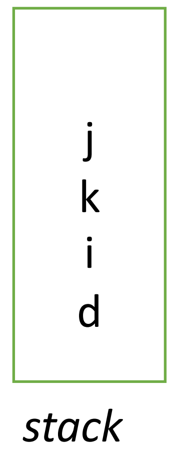


Order in which nodes are first visited: a, b, c, g, e, f, h, i, k

Order in which nodes are become dead ends: e, f, g, h, c, a

Step 18. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is 'k') which is not filled yellow. This gives us 'j'. Mark 'j' and add it to the top of the stack

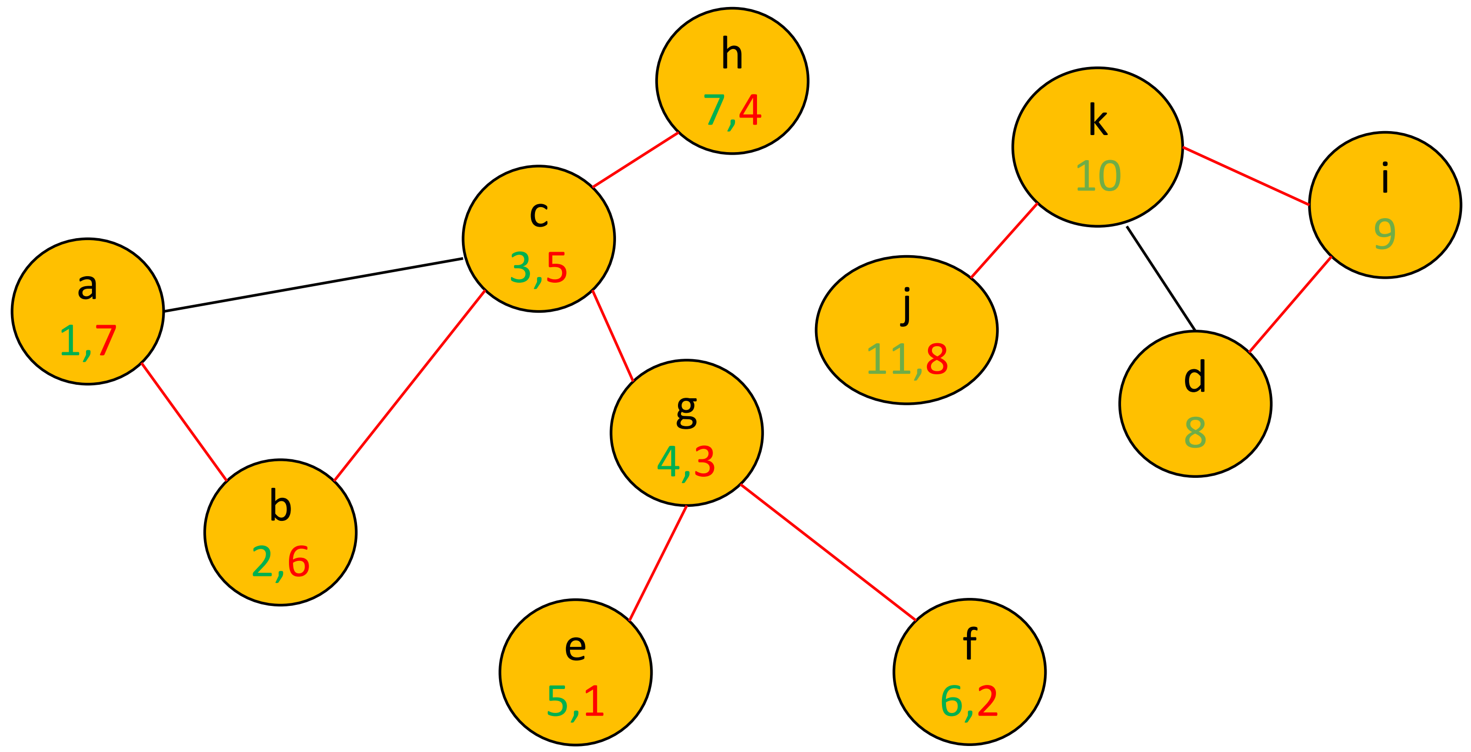


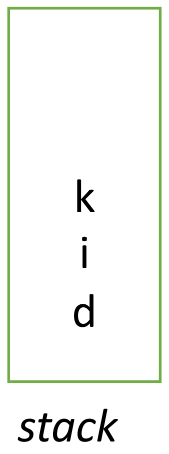


Order in which nodes are first visited: a, b, c, g, e, f, h, i, k, j

Order in which nodes are become dead ends: e, f, g, h, c, a

Step 19. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘j’) which is not filled yellow. But there’s no such vertex. Pop ‘j’ off the stack

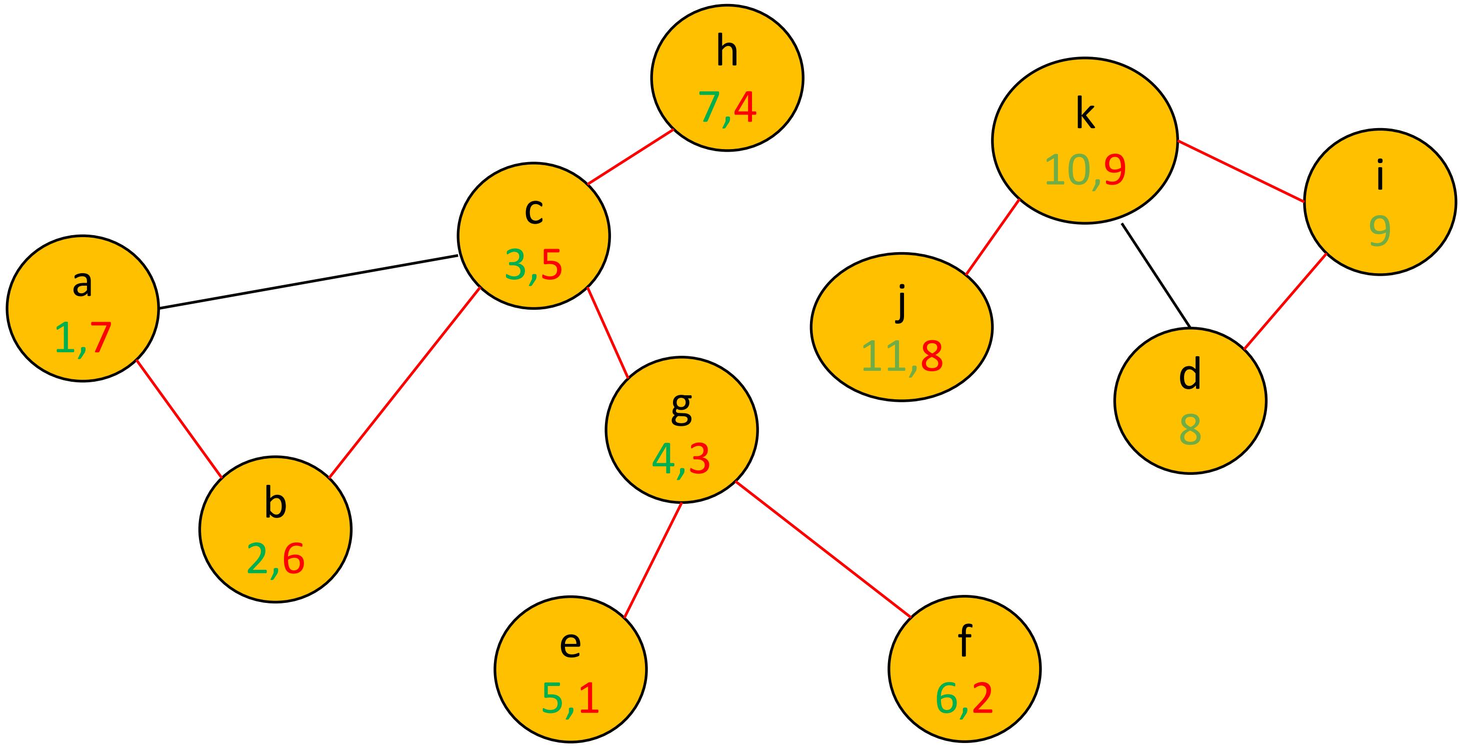


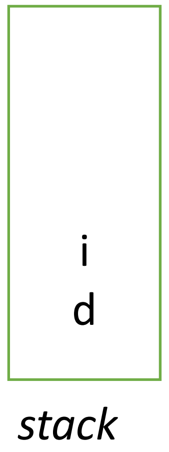


Order in which nodes are first visited: a, b, c, g, e, f, h, i, k, j

Order in which nodes are become dead ends: e, f, g, h, c, a, j

Step 19. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘k’) which is not filled yellow. But there’s no such vertex. Pop ‘k’ off the stack

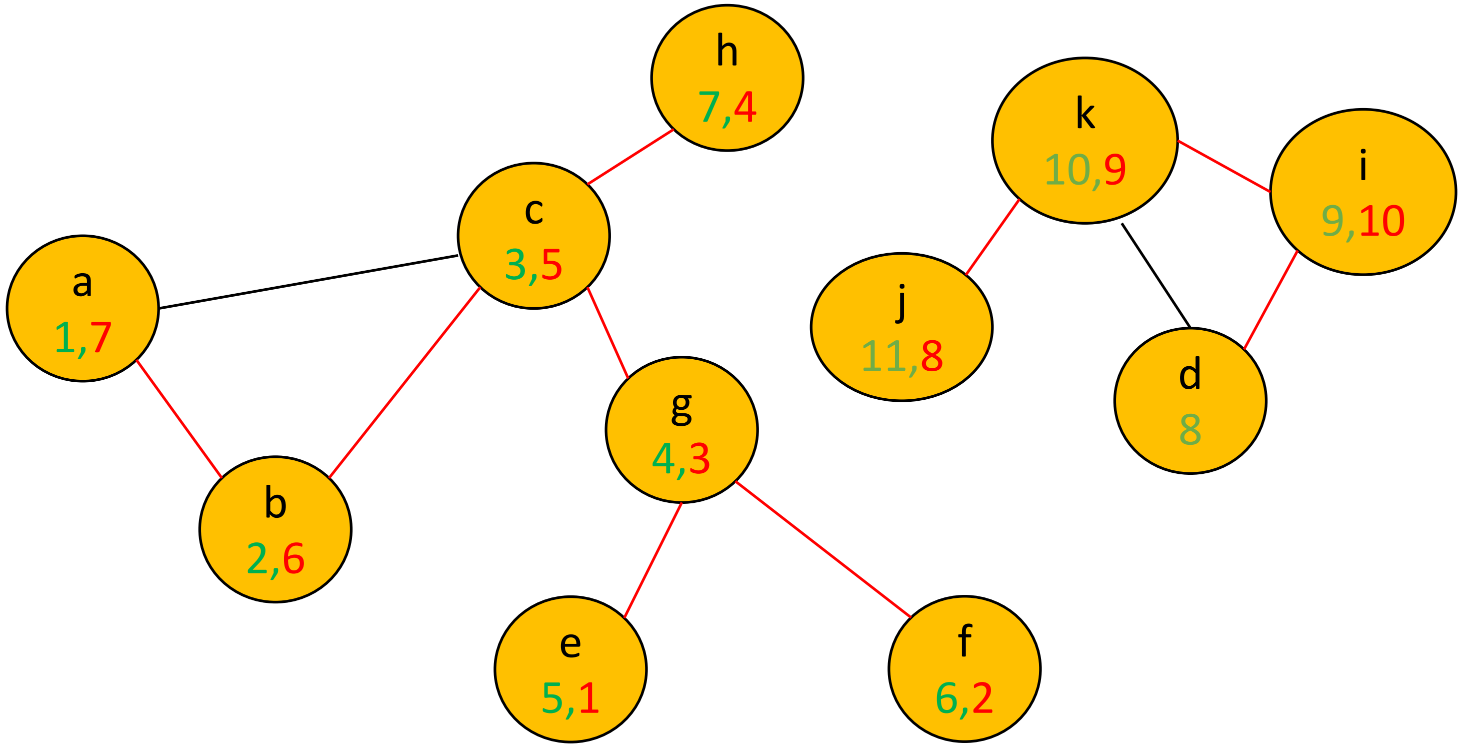


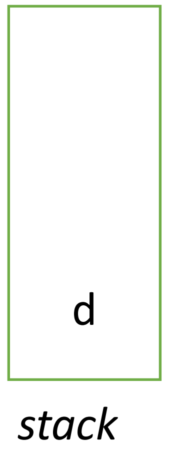


Order in which nodes are first visited: a, b, c, g, e, f, h, i, k, j

Order in which nodes are become dead ends: e, f, g, h, c, a, j, k

Step 20. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘i’) which is not filled yellow. But there’s no such vertex. Pop ‘i’ off the stack

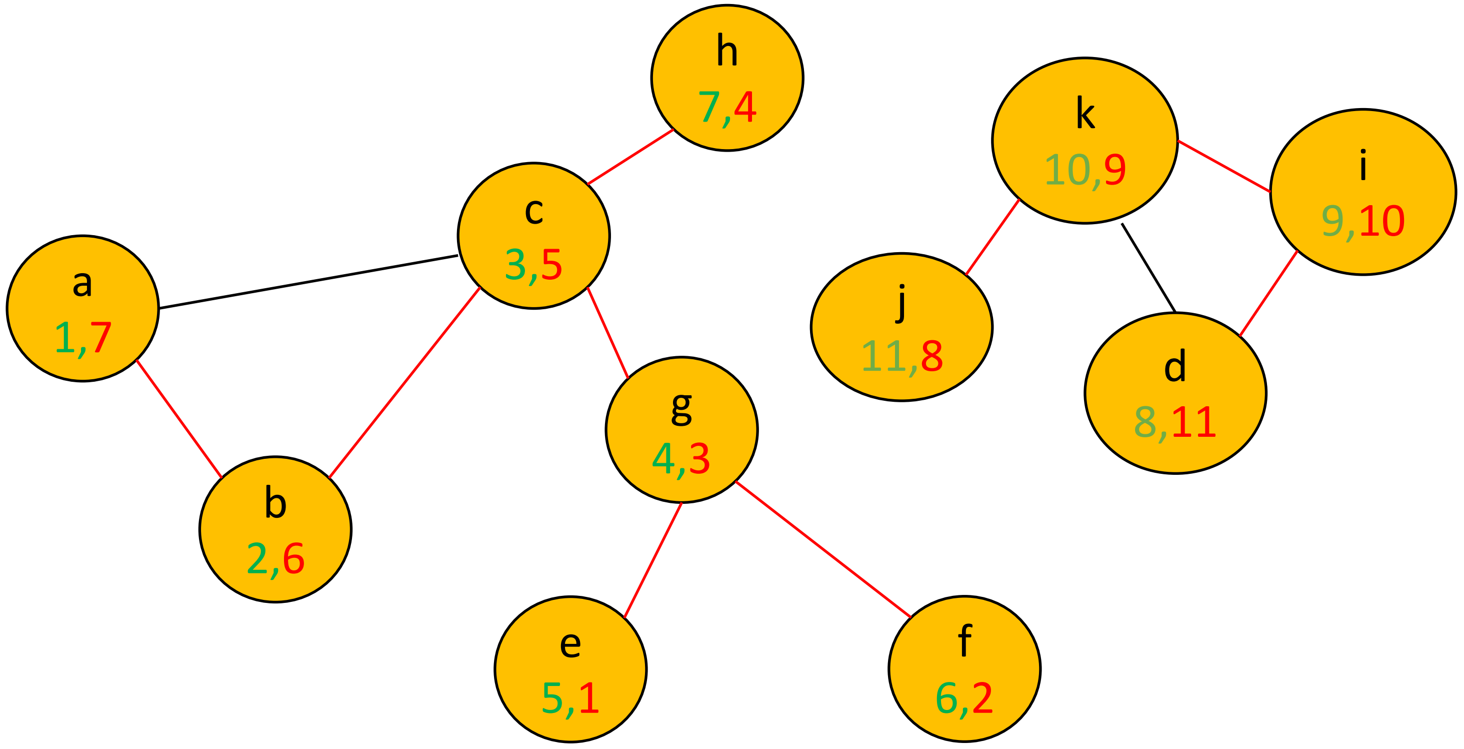


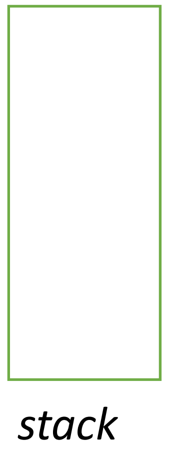


Order in which nodes are first visited: a, b, c, g, e, f, h, i, k, j

Order in which nodes are become dead ends: e, f, g, h, c, a, j, i

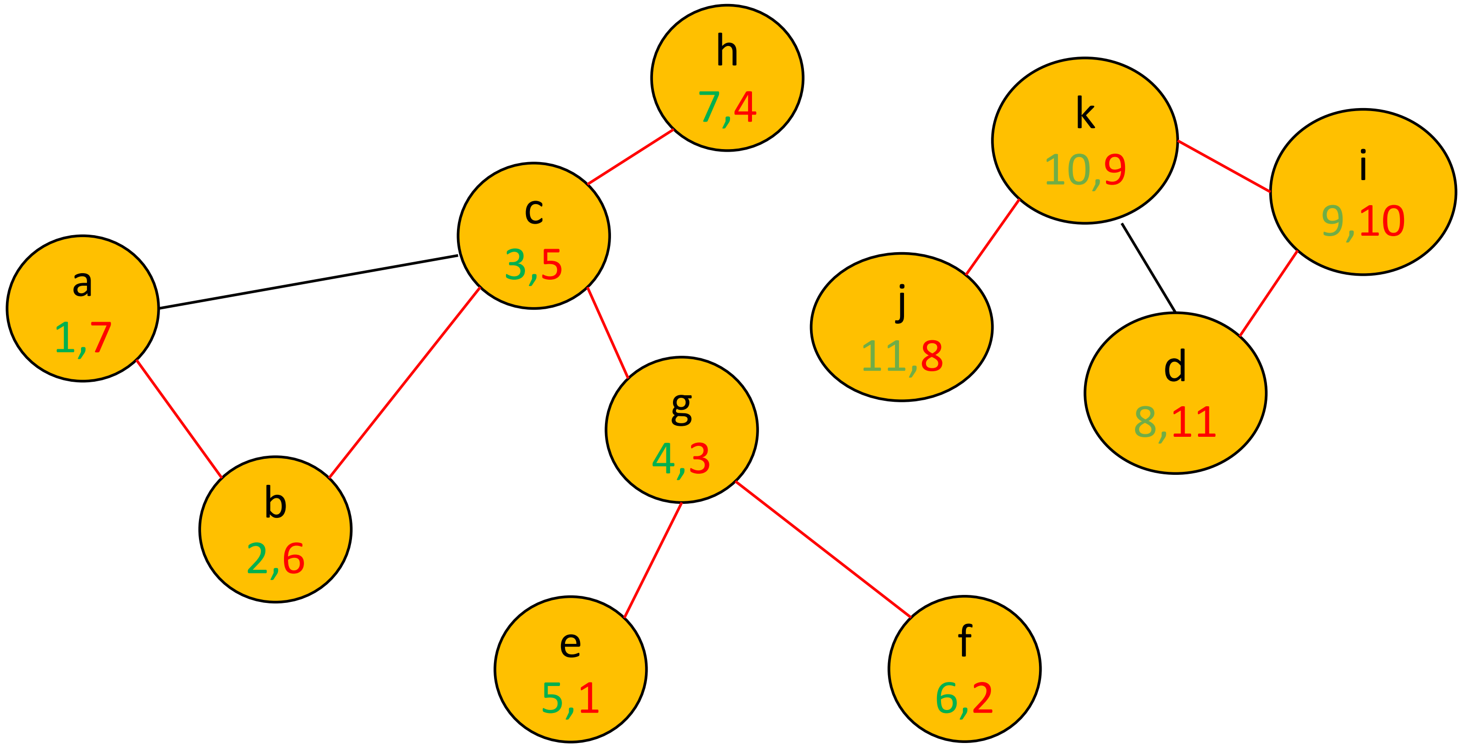
Step 20. Proceed to a vertex connected to the vertex in the top of the stack (currently, this is ‘d’) which is not filled yellow. But there’s no such vertex. Pop ‘d’ off the stack

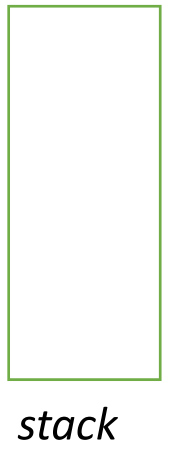




Order in which nodes are first visited: a, b, c, g, e, f, h, i, k, j
Order in which nodes are become dead ends: e, f, g, h, c, a, j, i, d

Step 21. Since our stack is empty, it means we have finished traversing one connected component of the graph. But there could be other connected components. To verify this, we check whether all the vertices are filled? The answer is YES. So we stop the algorithm.



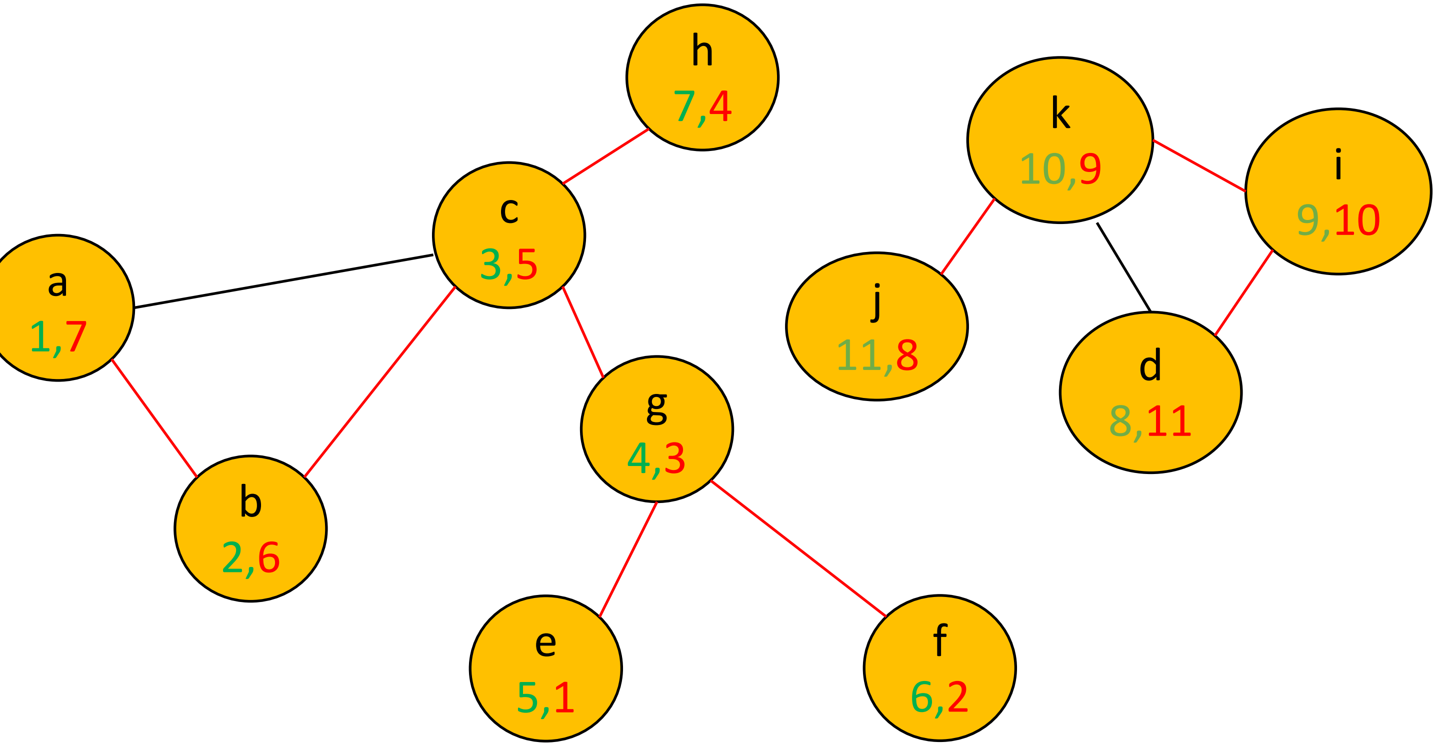


Order in which nodes are first visited: a, b, c, g, e, f, h, i, k, j

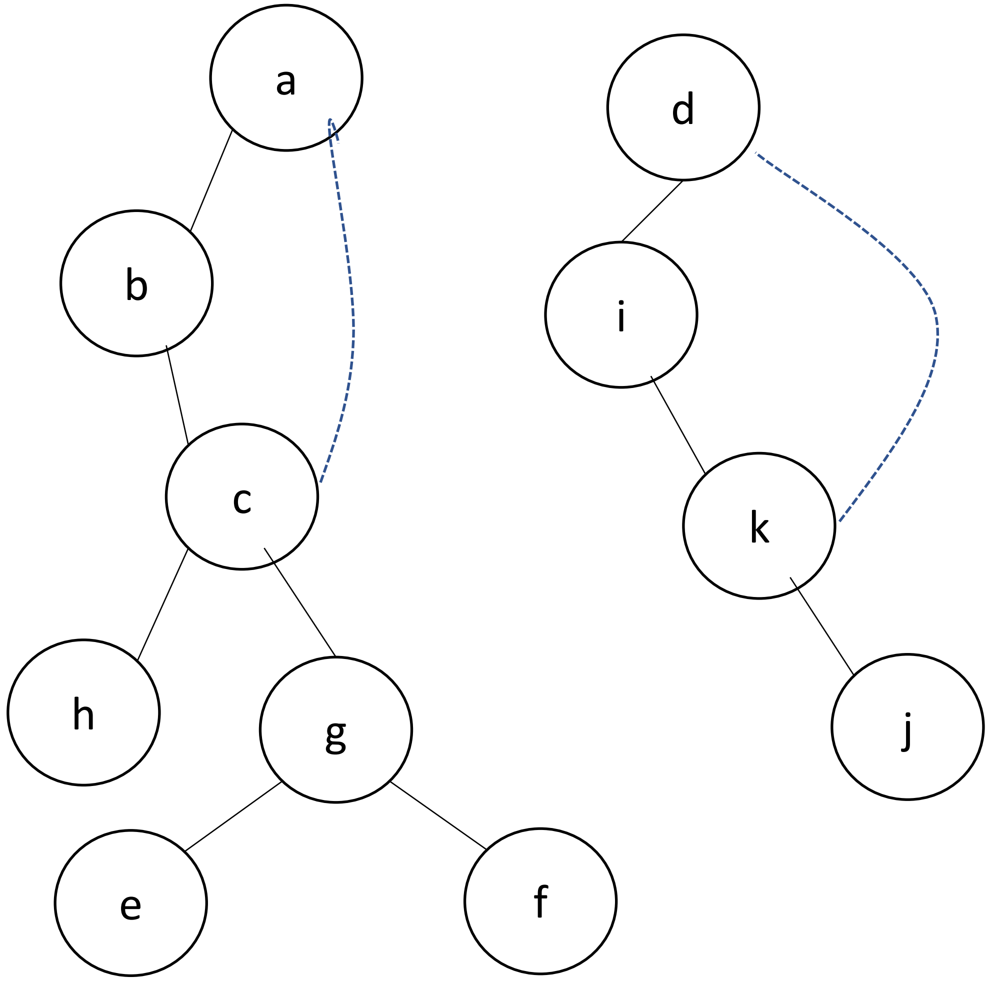
Order in which nodes are become dead ends: e, f, g, h, c, a, j, i, d

Now Let's draw the DFS traversal forest

The red edges will be tree edges in our traversal tree. The **black edges** will be back edges



Completely traversed graph



DFS forest

If you did not get bored till this point, I offer you my congratulations.