Algorithm to check whether a graph contains a cycle.

(we will assume a connected graph).

Idea: Start from any arbitrary vertex.

Perform a DFS starting from this vertex.
If you come back to an already visited node (which is not same as the node visited just before (*)), you have encountered a cycle.

For non-connected graphs, you can repeat this operation for every connected component.

PTO for pseudo code
DFS-cycle check \((G = (V, E))\)

1. \(v_1 = \) arbitrarily selected vertex from \(V\).
2. mark \(v_1\) as explored.
3. \(\text{previous-node}(v_1) = \text{NULL}\);
4. \(\text{push-stack}(v_1)\);

while (! empty-stack())

1. \(s = \text{peek stack}();\) // what's on the top of the stack?
2. \(t = \text{next neighbor of } s;\)
3. // if we had an adjacency list, it is easy to get the next neighbor. You can keep a counter or a pointer for each graph node
4. if \((t = \text{NULL})\) \(\text{popstack}();\)
5. else if \((t\) is explored)
6. \{ \(t = \text{previous-node}(s)\) continue;
7. \} // \(s-t-s\) is not a cycle!
8. else
9. \{ \(\text{return CYCLE-FOUND};\)
10. \} // the cycle can be obtained by popping
11. \} // the stack till you get a node equal to \(t\)
else
{
    previous-node (t) = s;
    mark t as explored;
    push-stack (t);
}
} // close while
}
Let's say we start from $V_1$.

Push $V_2$ onto stack
push $V_3$ onto stack
   " V4 " "
   " V5 " "
we can't push $V_3$ as it was already visited and is the previous vertex in our exploration)
push $V_6$ onto the stack
   " V7 " "

$V_7$ has neighbors $V_6$ and $V_3$. Both are explored. Out of this $V_6$ was the previous node of $V_7$, so we ignore it, but $V_3$ was already visited.

We just detected a cycle!
The cycle is obtained by popping elements $V_7$, $V_6$, $V_5$, $V_4$, $V_3$ from the cycle & adding $V_3$ before this list. The cycle is $V_3$ $V_7$ $V_6$ $V_5$ $V_4$ $V_3$. 