Problem 4

General idea:

Doing a DFS using stack, push the next node and the explore history to get to this node(the path to this node) together in the stack. Explore the graph, if the present pop node has a edge to the starting node, the present explore history add the starting point is the cycle. If the stack is empty, return false.

Prove it is liner time:

The algorithm is just doing DFS, using extra memory space to keep track of explore history to each node, so the time complexity is still O(E).

Algorithm:

```
Findcycle(graph,start):
    Stack.push( (start, None) );
    While true:
    {
        If stack is empty:
            Return false;
        now_node, now_path = stack.pop
        now_path append now_node
        for node in next_nodes:
            If node is start:
                Return (now_path append start)
            else:
                Stack.push( next_node, now_path)

    }
```

My algorithm return false with start point is 1.

However, as professor required, we need to provide Step by Step and Call Depth. I can provide the explore order using a stack, but cannot provide other information. So I conduct a DFS with recursion same with the first and second problem on the graph.

I think my algorithm can work according to the problem, but cannot provide some information.

In the perspective of finishing homework itself, I can modify my solution to this:

```
1.Perform DFS on graph
2.Get pre and post number
3.Classify the edges
4.If back edges not exisit:
5.    return false
6.    else if start node in them:
7.        get another node of the edge
8.        Doing DFS using stack:
9.            If next_node == another node:
10.               Return now_path append start
```

time complexity for step 1 and 2 is O(E)
time complexity for step 3 is O(E)
time complexity for step 4-6 is O(1)
time complexity for step 8-10 is O(E)
so the total is O(E+E+1+E) still O(E)

Explore order:
1 3 4 10 8 2 5 6 7 9

Value (pre, post):
vertex 1 :(1,16)
vertex 2 :(2,13)
vertex 3 :(4,7)
vertex 4 :(14,15)
vertex 5 :(17,20)
vertex 6 :(9,10)
vertex 7 :(3,12)
vertex 8 :(5,6)
vertex 9 :(18,19)

vertex 10 :(8,11)

edge classification:

forward is :

[(1, 2), (1, 4), (2, 7), (2, 8), (3, 8), (5, 9), (7, 3), (7, 10), (10, 6)]

back is :

[(6, 2), (8, 7)]

cross is

[(4, 7), (5, 10), (6, 8), (9, 1), (10, 3)]

Step by Step and Call Depth

now exploring 1

depth 1

visited[1] sets True

pre[1] sets 1

clock increase from 1 to 2

now exploring 2

depth 2

visited[2] sets True

pre[2] sets 2

clock increase from 2 to 3

now exploring 7

depth 3

visited[7] sets True

pre[7] sets 3

clock increase from 3 to 4

now exploring 3

depth 4

visited[3] sets True

pre[3] sets 4

clock increase from 4 to 5

now exploring 8

depth 5

```
visited[8] sets True
pre[8] sets 5
clock increase from 5 to 6
8 is already explored
post[8] sets 6
clock increase from 6 to 7
post[3] sets 7
clock increase from 7 to 8
now exploring 10
depth 4
visited[10] sets True
pre[10] sets 8
clock increase from 8 to 9
10 is already explored
now exploring 6
depth 5
visited[6] sets True
pre[6] sets 9
clock increase from 9 to 10
6 is already explored
6 is already explored
post[6] sets 10
clock increase from 10 to 11
post[10] sets 11
clock increase from 11 to 12
post[7] sets 12
clock increase from 12 to 13
2 is already explored
post[2] sets 13
clock increase from 13 to 14
now exploring 4
depth 2
visited[4] sets True
```

```
pre[4] sets 14
clock increase from 14 to 15
4 is already explored
post[4] sets 15
clock increase from 15 to 16
post[1] sets 16
clock increase from 16 to 17
now exploring 5
depth 1
visited[5] sets True
pre[5] sets 17
clock increase from 17 to 18
now exploring 9
depth 2
visited[9] sets True
pre[9] sets 18
clock increase from 18 to 19
9 is already explored
post[9] sets 19
clock increase from 19 to 20
5 is already explored
post[5] sets 20
```