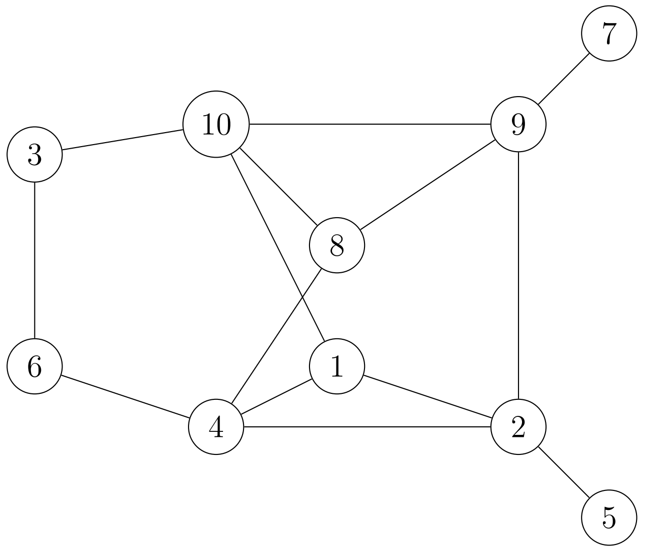
Given the adjacency list representation of graph G from tutorial. We'll trace through the steps of running DFS on G starting at node with value 1.

1	2, 4, 10
2	1, 4, 5, 9
3	6, 10
4	1, 2, 6, 8
5	2
6	3, 4
7	9
8	4, 9, 10
9	2, 7, 8, 10
10	1, 3, 8, 9

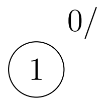


We'll have a stack on the side to help us keep track of edges we need to push/pop. The DFS-tree that starts at 1 will be drawn beside the stack. A node is drawn iff it's discovered (visited for the very first time) and will have an edge pointing toward it from whichever node that led to the discovery, that edge is called a tree-edge.

1.
```
Time = 0;
Stack.push(1, NULL); # push starting node
Stack.push((1, 2), (1, 4), (1, 10)); # 1's neighbours
```

(1, 10)
(1, 4)
(1, 2)
(1, NULL)

Stack



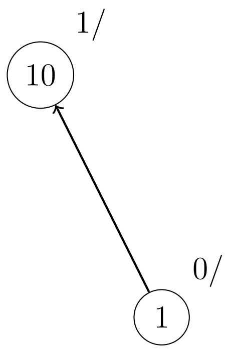
Note. We'll jot down the discovery time and finish time beside each node.

2.

```
Stack.pop(); # pops (1, 10)
Time = 1;
Stack.push((10, NULL), (10, 1), (10, 3), (10, 8), (10, 9));
```

(10, 9)
(10, 8)
(10, 3)
(10, 1)
(10, NULL)
~~(1, 10)~~
(1, 4)
(1, 2)
(1, NULL)

Stack

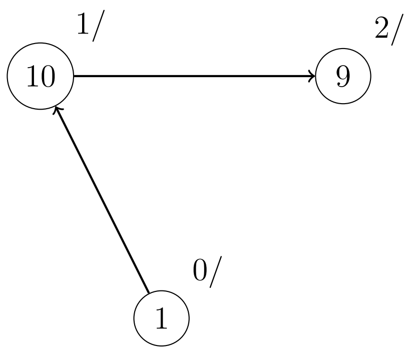


3.

```
Stack.pop(); # pops (10, 9)
Time = 2;
Stack.push((9, NULL), (9, 2), (9, 7), (9, 8), (9, 10))
```

(9, 10)
(9, 8)
(9, 7)
(9, 2)
(9, NULL)
~~(10, 9)~~
(10, 8)
(10, 3)
(10, 1)
(10, NULL)
(1, 4)
(1, 2)
(1, NULL)

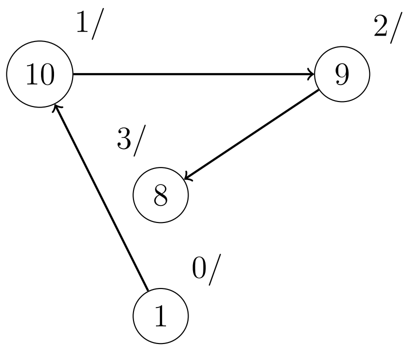
Stack



4.
```
Stack.pop(); # pops (9, 10)
             # Since 10 was already discovered, do nothing.
Stack.pop(); # pops (9, 8)
Time = 3;
Stack.push((8, NULL), (8, 4), (8, 9), (8, 10))
```

(8, 10)
(8, 9)
(8, 4)
(8, NULL)
~~(9, 10)~~
~~(9, 8)~~
(9, 7)
(9, 2)
(9, NULL)
(10, 8)
(10, 3)
(10, 1)
(10, NULL)
(1, 4)
(1, 2)
(1, NULL)

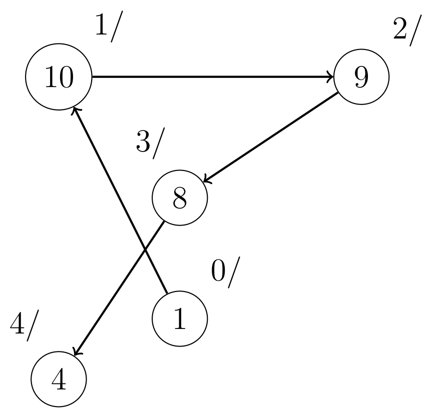
Stack



```
5.      Stack.pop(); # pops (8, 10), do nothing
                     # since 10 is 8's ancestor
                     this would be a back-edge in the DFS-tree.
                     # => G has 1 or more cycles.
        Stack.pop(); # pops (8, 9), do nothing
        Stack.pop(); # pops (8, 4)
        Time = 4;
        Stack.push((4, NULL), (4, 1), (4, 2), (4, 6), (4, 8))
```

(4, 8)
(4, 6)
(4, 2)
(4, 1)
(4, NULL)
~~(8, 10)~~
~~(8, 9)~~
~~(8, 4)~~
(8, NULL)
(9, 7)
(9, 2)
(9, NULL)
(10, 8)
(10, 3)
(10, 1)
(10, NULL)
(1, 4)
(1, 2)
(1, NULL)

Stack

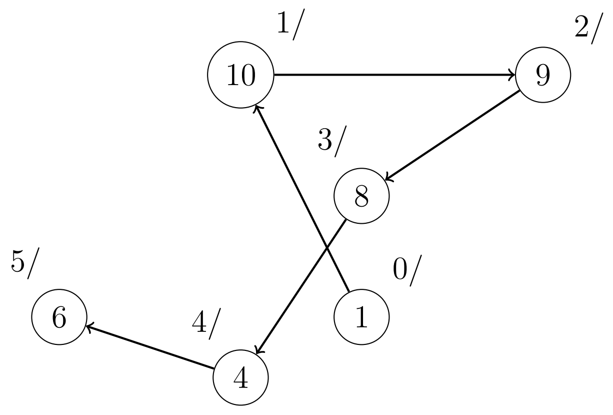


6.

```
Stack.pop(); # pops (4, 8), do nothing
Stack.pop(); # pops (4, 6)
Time = 5;
Stack.push((6, NULL), (6, 3), (6, 4))
```

(6, 4)
(6, 3)
(6, NULL)
~~(4, 8)~~
~~(4, 6)~~
(4, 2)
(4, 1)
(4, NULL)
(8, NULL)
(9, 7)
(9, 2)
(9, NULL)
(10, 8)
(10, 3)
(10, 1)
(10, NULL)
(1, 4)
(1, 2)
(1, NULL)

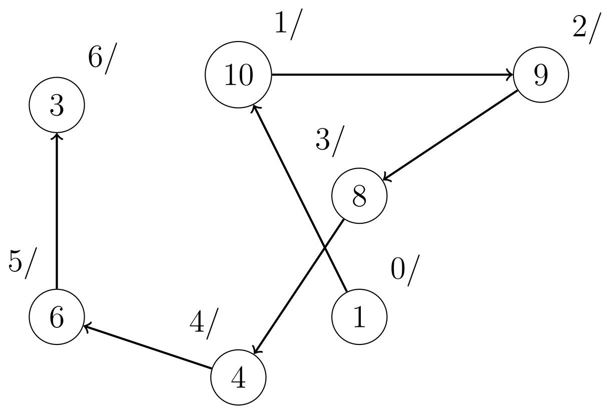
Stack



7.
```
Stack.pop(); # pops (6,4), do nothing
Stack.pop(); # pops (6,3)
Time = 6;
Stack.push((3, NULL), (3, 6), (3, 10));
```

(3, 10)
(3, 6)
(3, NULL)
~~(6, 4)~~
~~(6, 3)~~
(6, NULL)
(4, 2)
(4, 1)
(4, NULL)
(8, NULL)
(9, 7)
(9, 2)
(9, NULL)
(10, 8)
(10, 3)
(10, 1)
(10, NULL)
(1, 4)
(1, 2)
(1, NULL)

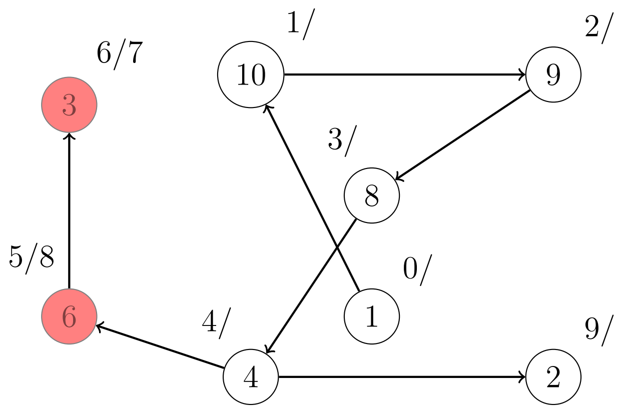
Stack



8.
```
Stack.pop(); # pops (3, 10), do nothing
Stack.pop(); # pops (3, 6), do nothing
Stack.pop(); # pops (3, NULL), finished
Time = 7;
Stack.pop(); # pops (6, NULL), finished
Time = 8;
Stack.pop(); # pops (4, 2)
Time = 9;
```

~~(3, 10)~~
~~(3, 6)~~
~~(3, NULL)~~
~~(6, 4)~~
~~(6, 3)~~
~~(6, NULL)~~
~~(4, 2)~~
(4, 1)
(4, NULL)
(8, NULL)
(9, 7)
(9, 2)
(9, NULL)
(10, 8)
(10, 3)
(10, 1)
(10, NULL)
(1, 4)
(1, 2)
(1, NULL)

Stack



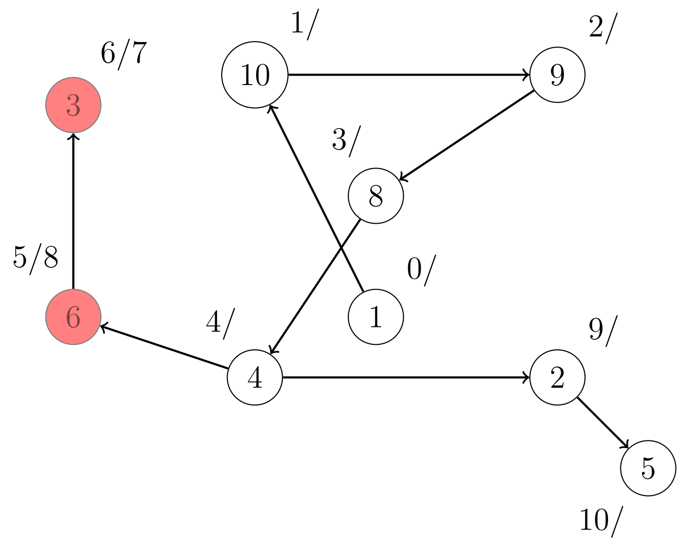
Note. We'll indicate a node's finished by colouring it in with red.

9.

```
Stack.push((2, NULL), (2, 1), (2, 4), (2, 5), (2, 9));
Stack.pop(); # pops (2, 9), do nothing
Stack.pop(); # pops (2, 5)
Time = 10;
```

~~(2, 9)~~
~~(2, 5)~~
(2, 4)
(2, 1)
(2, NULL)
(4, 1)
(4, NULL)
(8, NULL)
(9, 7)
(9, 2)
(9, NULL)
(10, 8)
(10, 3)
(10, 1)
(10, NULL)
(1, 4)
(1, 2)
(1, NULL)

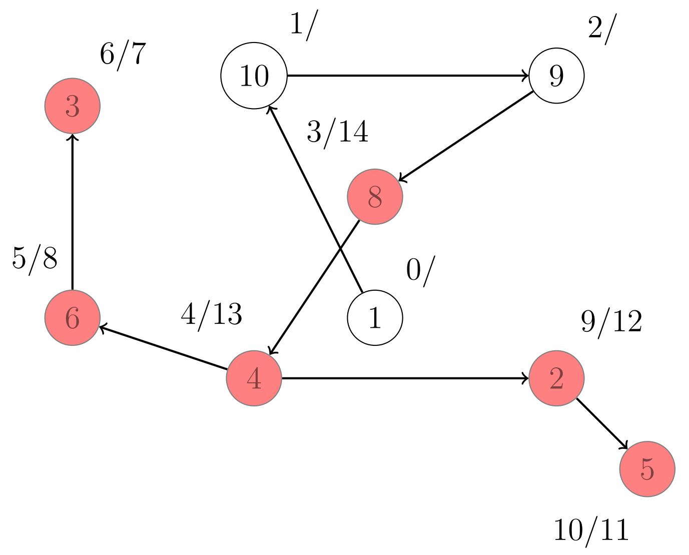
Stack



10.
```
Stack.push((5, NULL), (5, 2))
Stack.pop(); # pop (5, 2), do nothing
Stack.pop(); # pop (5, NULL), finished
Time = 11;
Stack.pop(); # pop (2, 4), do nothing
Stack.pop(); # pop (2, 1), do nothing
Stack.pop(); # pop (2, NULL), finished
Time = 12;
Stack.pop(); # pop (4, 1), do nothing
Stack.pop(); # pop (4, NULL), finished
Time = 13
Stack.pop(); # pop (8, NULL), finished
Time = 14
```

~~(5, 2)~~
~~(5, NULL)~~
~~(2, 4)~~
~~(2, 1)~~
~~(2, NULL)~~
~~(4, 1)~~
~~(4, NULL)~~
~~(8, NULL)~~
(9, 7)
(9, 2)
(9, NULL)
(10, 8)
(10, 3)
(10, 1)
(10, NULL)
(1, 4)
(1, 2)
(1, NULL)

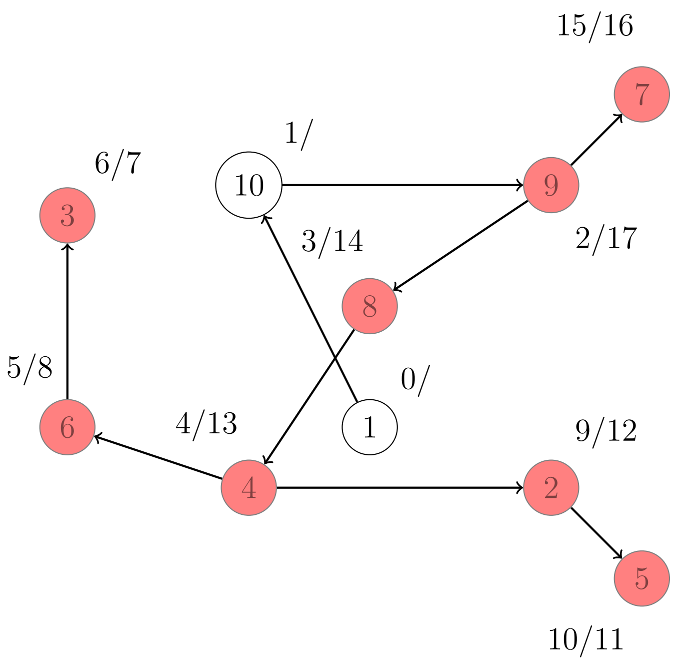
Stack



11.
```
Stack.pop() # pops (9, 7)
Time = 15;
Stack.push((7, NULL), (7, 9));
Stack.pop(); # pops (7, 9), do nothing
Stack.pop(); # pops (7, NULL), finished
Time = 16;
Stack.pop(); # pops (9, 2), do nothing
Stack.pop(); # pops (9, NULL), finished
Time = 17;
```

~~(7, 9)~~
~~(7, NULL)~~
~~(9, 7)~~
~~(9, 2)~~
~~(9, NULL)~~
(10, 8)
(10, 3)
(10, 1)
(10, NULL)
(1, 4)
(1, 2)
(1, NULL)

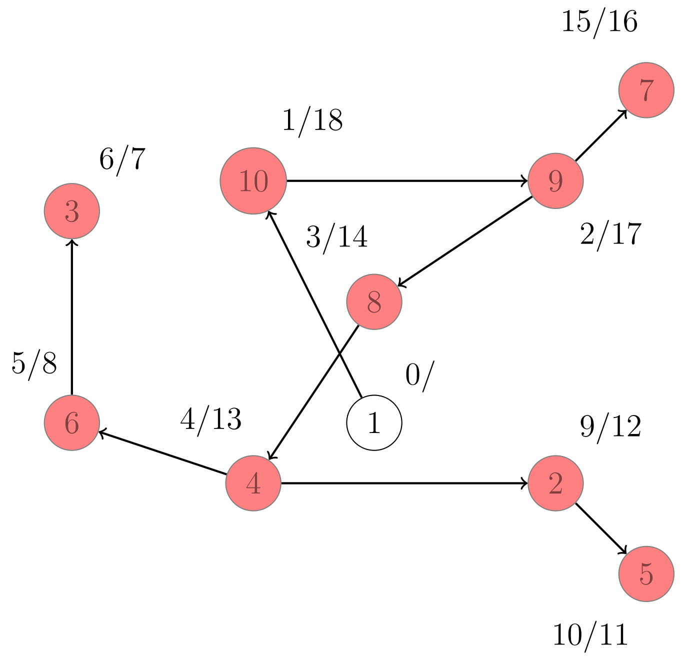
Stack



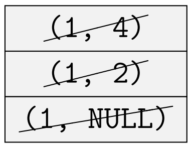
12.
```
Stack.pop() # pops (10, 8), do nothing
Stack.pop() # pops (10, 3), do nothing
Stack.pop() # pops (10, 1), do nothing
Stack.pop() # pops (10, NULL), finished
Time = 18;
```

~~(10, 8)~~
~~(10, 3)~~
~~(10, 1)~~
~~(10, NULL)~~
(1, 4)
(1, 2)
(1, NULL)

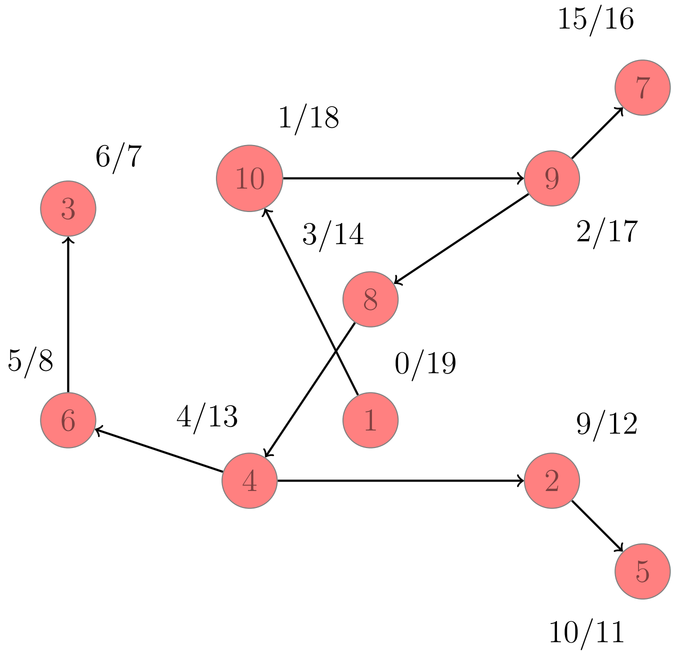
Stack



13.
```
Stack.pop() # pops (1, 4), do nothing
Stack.pop() # pops (1, 2), do nothing
Stack.pop() # pops (1, NULL), finished
Time = 19;
```



Stack



15/16
7
1/18
10
9
6/7
3
3/14
2/17
8
5/8
0/19
6
4/13
1
9/12
4
2
5
10/11