① Algorithm connectedComponents(G)
    return DFS(G)

Algorithm initResult(G)
    S := new Sequence

Algorithm postComponentVPost(G, v)
    S.insertLast(v)
Algorithm result(G)
    return S

```
Algorithm findPath(G, s, L)
    start ← s
    dest ← L
    return BFS(G)
Algorithm initResult(G)
    S new stack
    Path new list stack
Algorithm isNextComponent(G, v)
    return start == v
Algorithm postDiscEdgeVisit(G, v, e, w)
    if S.top() != dest then
        S.push(e)
        S.push(w)
Algorithm finishBFScomponent(G, s)
    top := S.top()
    if top = dest then
        path.insertLast()
    While !S.isEmpty do
        vertex := S.pop()
        edge := S.pop()
        if getLabel(edge) = Discovery then
            path.push(vertex)
            path.push(edge)
    Path.push(start)
Algorithm result(G)
    if path.size = 0 then return NO_SUCH_PATH
    else path.elements()
```

CYCLE BFS

```
Algorithm findCycle (G)
    return BFS (G)
Algorithm initResult(G)
    cycleFound := false
    cycle := new sequence
```

```
Algorithm result(G)
  if cycleFound then
    return cycle
  return NO_SUCH_CYCLE
```

```
Algorithm preDiscEdgeVisit ( G, v, e, w)
    setParent (w, e)
```

```
Algorithm crossEdgeVisit (G, v, e, w)
    if !cycleFound then
        buildCycle( G, v, e, w)
```

```
Algorithm buildCycle ( G, v, e, w)
    cycle.insertLast (w)
    left := w
    right := w
    cycle.insertFirst (e)
    cycle.insertFirst (v)
    while left ≠ right do
        e1 := getParent (left)
        if e1 ≠ ∅ then
            left := G.opposite (e1, left)
            cycle.insertFirst (e1)
            cycle.insertFirst (left)
        e2 := getParent (right)
        if e2 ≠ ∅ then
            right := G.opposite (e2, right)
            cycle.insertLast (e2)
            cycle.insertLast (right)
    cycleFound := true
```

```
Algorithm  counterMet(G)
    return BFS(G)
Algorithm  initResult(G)
    counter := 0
Algorithm  posComponentVisit(G, v)
    counter++,
Algorithm  startBFSComponent(G, s)
    setLabel(s, counter)
Algorithm  postDiscEdgeVisit(G, v, e, w)
    setLabel(w, counter)
Algorithm  result(G)
    return G
```