# Assignment 13

1. Using the DFS Template Method Pattern algorithm given in the lecture notes, override the appropriate methods so this algorithm computes the connected components of a graph G. Your method should return a sequence of vertices, 1 representative from each connected component.

Solution

```
Algorithm initResult(G)
components ← new empty List

Algorithm preComponentVisit(G, v)
components.insertLast(v)

Algorithm result(G)
return components
```

2. a. Modify the breadth-first search algorithm so it can be used as a Template Method Pattern.

```
Algorithm BFS(G)
  initResult(G)
  for each vertex u in G
    setLabel(u, UNEXPLORED)
    postInitVertex(G, u)

  for each edge e in G
    setLabel(e, UNEXPLORED)
    postInitEdge(G, e)

  for each vertex v in G
    if isNextComponent(G, v)
      preComponentVisit(G, v)
      BFScomponent(G, v)
      postComponentVisit(G, v)

  return result(G)


Algorithm BFScomponent(G, s)
  Q ← new empty Queue
  Q.enqueue(s)
  setLabel(s, VISITED)
  beginVertexVisit(G, s)

  while Q is not empty
    v ← Q.dequeue()
    for each edge e in incidentEdges(v)
      preEdgeVisit(G, v, e)
      if getLabel(e) == UNEXPLORED
        w ← opposite(v, e)
        edgeVisit(G, v, e, w)
        if getLabel(w) == UNEXPLORED
          setLabel(e, DISCOVERY)
          preDiscoveryVisit(G, v, e, w)
          setLabel(w, VISITED)
          Q.enqueue(w)
          postDiscoveryVisit(G, v, e, w)
        else
          setLabel(e, CROSS)
          crossEdgeVisit(G, v, e, w)
      postEdgeVisit(G, v, e)
    finishVertexVisit(G, v)
```

b. Write a pseudo code function findPath(G, u, v) that uses your Template Method from (a) to find a path in G between vertices u and v with the minimum number of edges, or report that no such path exists. Hint: Override the appropriate methods so that given two vertices u and v of G, your call to BFS finds and returns a Sequence containing the path between u and v.

```
Method: initResult(G)
  path ← new empty sequence

Method: postInitVertex(v)
  setParent(v, null)

Method: preDiscoveryVisit(G, v, e, w)
  setParent(w, e)

Method: beginVertexVisit(G, v)
  if v == dest then
    path ← buildPath(G, v)

Method: isNextComponent(G, v)
  return getLabel(v) == UNEXPLORED and v == start

Method: result(G)
  if path is empty then
    return "No path exists"
  else
    return path
```

c. Write a pseudo code function findCycle(G) that uses your Template Method from (a) to find a simple cycle in a graph G (any cycle, not all cycles). That is, override the appropriate methods so your solution finds a cycle in G. You are to return a Sequence containing the cycle.

```
Variables (subclass fields):
  parent: Map<Vertex, Edge>
  cycle: Sequence
  cycleFound: Boolean

Method: findCycle(G)
  return BFS(G)

Method: initResult(G)
  parent ← empty map
  cycle ← empty sequence
  cycleFound ← false

Method: postInitVertex(v)
  parent.insertItem(v, null)

Method: preDiscoveryVisit(G, v, e, w)
    parent.insertItem(w, e)

Method: crossEdgeVisit(G, v, e, w)
  if not cycleFound then
    if w ≠ parent of v then
      cycle ← buildCycle(G, v, w, e)
      cycleFound ← true

Method: isNextComponent(G, v)
  return not cycleFound and getLabel(v) == UNEXPLORED

Method: result(G)
  if cycleFound then
    return cycle
  else
    return "No cycle found"
```

d. Can the template version of DFS be used to find the path between two vertices with the minimum number of edges? Briefly explain why or why not.

No, DFS does not guarantee the path with the minimum number of edges because it explores paths deeply and doesn't consider shortest distance.
Only BFS guarantees the shortest path in unweighted graphs.

4. Based on either the DFS or the BFS template method algorithms, write the overriding methods so that all nodes in each connected component of a graph G are labeled with a sequence number, i.e., each vertex in a component would be labeled with the same number. For example, each node in the first connected component would be labeled with a 0, each node in the second connected component would be labeled with a 1, etc.

```
Method: initResult(G)
  componentNumber ← 0
  label ← empty map

Method: preComponentVisit(G, v)
 found → label will use current componentNumber

Method: beginVertexVisit(G, v)
label.insertItem(v, componentNumber)

Method: postComponentVisit(G, v)
  componentNumber ← componentNumber + 1

Method: result(G)
  return label
```