Assignment 13

▼ Question 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.

```
Algorithm DFS(G)
   Input graph G
   Output labeling of the edges of G    as discovery edges and back edges
   initResult(G)
   for all u ∈ G.vertices() do
      setLabel(u, UNEXPLORED)
      postInitVertex(G, u)

   for all e ∈ G.edges() do
      setLabel(e, UNEXPLORED)
      postInitEdge(G, e)

   for all v ∈ G.vertices() do
      if getLabel(v) = UNEXPLORED
         preComponentVisit(G,v)
         DFScomponent(G, v)
         postComponentVisit(G, v)

   return result(G)
```

```
Algorithm DFScomponent(G, v)
   Input graph G and a start vertex v of G
   Output labeling of the edges of G in the connected component of v as discovery edges and back edges

   setLabel(v, VISITED)
   startVertexVisit(G,v,e)
   for all e ∈ G.incidentEdges(v) do
      preEdgeVisit(G,v,e)
      if getLabel(e) = UNEXPLORED
         w ∈ G.opposite(v,e)
         edgeVisit(G,v,e,w)
         if getLabel(w) = UNEXPLORED
            setLabel(e, DISCOVERY)
            preDiscoveryVisit(G,v,e,w)
            DFScomponent(G, w)
            postDiscoveryVisit(G,v,e,w)
         else
            setLabel(e, BACK)
            backEdgeVisit(G,v,e,w)
      finishVertexVisit()
```

```
Algorithm    initResult(G)
   cc = new empty List()

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

Algorithm result(G)
   return cc
```

▼ Question 2.a

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

```
Algorithm BFS(G)

   for all u ∈ G.vertices() do
      setLabel(u, UNEXPLORED)
      postInitVertex(G,u)

   for all e ∈ G.edges() do
      setLabel(e, UNEXPLORED)
      postInitEdge(G,e)

   for all v ∈ G.vertices() do
      if getLabel(v) = UNEXPLORED then
         preComponentVisit(G,v)
         BFScomponent(G, v)
         postComponentVisit(G,v)
```

```
Algorithm BFScomponent(G, s)
   Q ← new empty Queue
   Q.enqueue(s)
   startBFScomponent(G,s)
   setLabel(s, VISITED)
   while Q.size() > 0 do
      v ← Q.dequeue ()
      preVertexVisit(G,v)
      for all e ∈ G.incidentEdges(v) do
         if getLabel(e) = UNEXPLORED then
            w ← G.opposite(v,e)
            edgeVisit(G,v,e,w)
            if getLabel(w) = UNEXPLORED then
               preDiscoveryEdgeVisit(G,v,e,w)
               setLabel(e, DISCOVERY)
               setLabel(w, VISITED)
               Q.enqueue(w)
               postDiscoveryEdgeVisit(G,v,e,w)
            else
               setLabel(e, CROSS)
```

```
            crossEdgeVisit(G,v,e,w)
    finishBFScomponent(G,s)
```

▼ Question 2.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.

```
Algorithm BFSFindPath(G, a, b)
    D ← new Dictionary(Hash Table)
    found ← false

    for all u ∈ G.vertices() do
        setLabel(u, UNEXPLORED)
        postInitVertex(G, u) {
            D.set(u, null)
        }()

    for all e ∈ G.edges() do
        setLabel(e, UNEXPLORED)
        postInitEdge(G, e)

    preComponentVisit(G, a)
    BFScomponent(G, a, b)
    postComponentVisit(G, b)
    if found = true then
        S ← empty Sequence
        p ← D.get(b)
        S.insertFirst(p)

        while p != a do
            p ← D.get(p)
            S.insertFirst(p)
```

```
Algorithm BFScomponent(G, cur, target, D, found)
    Q ← new empty Queue
    Q.enqueue(cur)
    startBFScomponent(G, cur)
    setLabel(cur, VISITED)
    while Q.size() > 0 do
        from ← Q.dequeue()
        preVertexVisit(G, from) {
            if from = target then
                found ← true
        }()
        for all e ∈ G.incidentEdges(from) do
            if getLabel(e) = UNEXPLORED then
                toV ← G.opposite(fromV, e)
                edgeVisit(G, fromV, e, toV)
```

```
            if getLabel(toV) = UNEXPLORED then
                preDiscoveryEdgeVisit(G, fromV, e, toV) {
                    if D.get(toV) = null
                        D.set(toV, e)
                        D.set(e, fromV)
                }
                setLabel(e, DISCOVERY)
                setLabel(to, VISITED)
                Q.enqueue(toV)
                postDiscoveryEdgeVisit(G,fromV,e,toV) {
                    if found = true
                        return
                }()
            else
                setLabel(e, CROSS)
                crossEdgeVisit(G,from,e,to)
    finishBFScomponent(G,s)
```

▼ Question 2.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.

```
Algorithm BFSFindCycle(G)
    D ← new Dictionary(Hash Table)
    found ← false

    for all u ∈ G.vertices() do
        setLabel(u, UNEXPLORED)
        postInitVertex(G, u) {
            D.set(u, null)
        }()

    for all e ∈ G.edges() do
        setLabel(e, UNEXPLORED)
        postInitEdge(G, e)


    for all v ∈ G.vertices() do
        if getLabel(v) = UNEXPLORED then
            preComponentVisit(G,v)
            (from, edge, to) ← BFScomponent(G, v)
            postComponentVisit(G,v)

            if from != null ^ edge != null ^ to != null ^  then
                S ← new empty Sequence
                p1 ← D.get(from)
                p2 ← D.get(to)
```

```
            S.insertLast(edge)

            while p1 != v do
               p1 ← D.get(p1)
               S.insertLast(p1)

            while p2 != v do
               p2 ← D.get(p2)
               S.insertFirst(p2)

            return S

   return new empty Sequence
```

```
Algorithm BFScomponent(G, cur, target, D, found)
   Q ← new empty Queue
   Q.enqueue(cur)
   startBFScomponent(G, cur)
   setLabel(cur, VISITED)
   while Q.size() > 0 do
      from ← Q.dequeue()
      preVertexVisit(G, from)
      for all e ∈ G.incidentEdges(from) do
         if getLabel(e) = UNEXPLORED then
            toV ← G.opposite(fromV, e)
            edgeVisit(G, fromV, e, toV)
            if getLabel(toV) = UNEXPLORED then
               preDiscoveryEdgeVisit(G, fromV, e, toV) {
                  if D.get(toV) = null
                     D.set(toV, e)
                     D.set(e, fromV)
               }
               setLabel(e, DISCOVERY)
               setLabel(to, VISITED)
               Q.enqueue(toV)
               postDiscoveryEdgeVisit(G, fromV, e, toV)
            else
               setLabel(e, CROSS)
               crossEdgeVisit(G, fromV, e, toV) {
                  return (fromV, e, toV)
               }()
   finishBFScomponent(G,s) {
      return (null, null, null)
   }
```

▼ Question 2.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.

DFS explores as deeply as possible before backtracking. It does **not prioritize the shortest path**.

▼ Question 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.

```
Algorithm BFS(G)

   for all u ∈ G.vertices() do
      setLabel(u, UNEXPLORED)
      postInitVertex(G,u)

   for all e ∈ G.edges() do
      setLabel(e, UNEXPLORED)
      postInitEdge(G,e)

   count ← 0
   for all v ∈ G.vertices() do
      if getLabel(v) = UNEXPLORED then
         preComponentVisit(G,v) {
            count++
         }
         BFScomponent(G, v, count)
         postComponentVisit(G,v)
```

```
Algorithm BFScomponent(G, s, label)
   Q ← new empty Queue
   Q.enqueue(s)
   startBFScomponent(G,s) {
      setLabel(s, label)
   }
   //setLabel(s, VISITED)
   while Q.size() > 0 do
      v ← Q.dequeue ()
      preVertexVisit(G,v)
      for all e ∈ G.incidentEdges(v) do
         if getLabel(e) = UNEXPLORED then
            w ← G.opposite(v,e)
            edgeVisit(G,v,e,w)
            if getLabel(w) = UNEXPLORED then
               preDiscoveryEdgeVisit(G,v,e,w)
               setLabel(e, DISCOVERY)
               setLabel(label)
               //setLabel(w, VISITED)
               Q.enqueue(w)
               postDiscoveryEdgeVisit(G,v,e,w) {
                  setLabel(w, label)
               }
            else
               setLabel(e, CROSS)
```

```
            crossEdgeVisit(G,v,e,w)
    finishBFScomponent(G,s)
```