Assignement 13

---

1. Compute Connected Components Using DFS

To compute the connected components of a graph, we use the DFS algorithm.

We need to override the DFS methods to visit all vertices of each connected component, mark them, and return one representative vertex from each connected component.

```
Algorithm computeConnectedComponents(G):
  Input: Graph G
  Output: A sequence of representative vertices from each connected component
  connectedComponents = []
  for u in G.vertices() do
    setLabel(u, UNEXPLORED)
  for v in G.vertices() do
    if getLabel(v) == UNEXPLORED then
      representative = DFScomponent(G, v)
      connectedComponents.append(representative)
  return connectedComponents

Algorithm DFScomponent(G, v):
  Input: Graph G, starting vertex v
  Output: The representative vertex of the connected component
  setLabel(v, VISITED)
  representative = v
```

```
  for each edge e in G.incidentEdges(v) do
    w = G.opposite(v, e)
    if getLabel(w) == UNEXPLORED then
      DFScomponent(G, w)
  return representative
```

2. Modify BFS and Find Paths and Cycles

(a) Modify BFS to Use Template Method Pattern

```
Algorithm BFScomponent(G, v):
  Input: Graph G, starting vertex v
  Output: Visits all vertices in the connected component of v
  queue = new Queue()
  queue.enqueue(v)
  setLabel(v, VISITED)
  while queue is not empty do
    u = queue.dequeue()
    processVertex(u)
    for e in G.incidentEdges(u) do
      w = G.opposite(u, e)
      if getLabel(w) == UNEXPLORED then
        setLabel(w, VISITED)
        processEdge(e)
        queue.enqueue(w)
```

=>This template can be reused and customized by overriding the `processVertex` and `processEdge` methods to perform tasks like pathfinding or cycle detection

(b) Find Shortest Path Between Two Vertices

To find the shortest path between two vertices using the BFS template, we can override the methods to keep track of the path as we explore the graph.

*Pseudo-code: Shortest Path using BFS

```
Algorithm findPath(G, u, v):
  Input: Graph G, start vertex u, target vertex v
  Output: A sequence of vertices representing the path from u to v, or report no path exists
  path = new Sequence()
  parent = new Map()
  queue = new Queue()
  queue.enqueue(u)
  setLabel(u, VISITED)
  parent[u] = null

  while queue is not empty do
    x = queue.dequeue()
    if x == v then
      while x != null do
        path.prepend(x)
        x = parent[x]
      return path
    for e in G.incidentEdges(x) do
      y = G.opposite(x, e)
      if getLabel(y) == UNEXPLORED then
        setLabel(y, VISITED)
        parent[y] = x
        queue.enqueue(y)
  return "No path exists"
```

(c) Find a Cycle using BFS

To detect a cycle, we modify BFS to look for a back edge (an edge leading to a previously visited vertex that is not its parent in the BFS tree).

Pseudo-code: Cycle Detection using BFS

```
Algorithm findCycle(G):
  Input: Graph G
  Output: A sequence of vertices representing a cycle, or report no cycle found
  parent = new Map()
  cycle = new Sequence()
  for u in G.vertices() do
    if getLabel(u) == UNEXPLORED then
      if BFSCycle(G, u, parent, cycle) == true then
        return cycle
  return "No cycle found"

Algorithm BFSCycle(G, v, parent, cycle):
  queue = new Queue()
  queue.enqueue(v)
  setLabel(v, VISITED)
  parent[v] = null
  while queue is not empty do
    x = queue.dequeue()
    for each edge e in G.incidentEdges(x) do
      y = G.opposite(x, e)
      if getLabel(y) == UNEXPLORED then
        setLabel(y, VISITED)
        parent[y] = x
```

```
        queue.enqueue(y)
      else if parent[x] != y then
        cycle.prepend(y)
        cycle.prepend(x)
        return true
  return false
```

(d) Can DFS be used to find the shortest path?

* No, DFS cannot be used to find the path between two vertices with the minimum number of edges because DFS explores as deeply as possible before backtracking.

This can lead to a non-optimal path, as DFS doesn't prioritize paths with fewer edges, unlike BFS, which explores level by level, ensuring the shortest path in terms of the number of edges.

4. Labeling Nodes by Connected Component

For this task, we use a DFS or BFS to label each vertex in the graph by the connected component it belongs to. Each connected component will have a unique label.

Pseudo-code: Labeling Connected Components

```
Algorithm labelConnectedComponents(G):
  Input: Graph G
  Output: Each vertex labeled with its connected component number
  componentNumber = 0
  for u in G.vertices() do
    if getLabel(u) == UNEXPLORED then
      BFSLabelComponent(G, u, componentNumber)
      componentNumber = componentNumber + 1
```

```
Algorithm BFSLabelComponent(G, v, componentNumber):
  queue = new Queue()
  queue.enqueue(v)
  setLabel(v, componentNumber)
  while queue is not empty do
    u = queue.dequeue()
    for e in G.incidentEdges(u) do
      w = G.opposite(u, e)
      if getLabel(w) == UNEXPLORED then
        setLabel(w, componentNumber)
        queue.enqueue(w)
```