Assignment 13

The following are to be written up and turned in separately from the rest of the homework.

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 isConnected(G)  Return DFS(G)  Algorithm initResult(G)  compCount := 0  Algorithm preComponentVisit(G, v)  compCount ++  Algorithm result(G)  if compCount = 1 then  return True  else  return False |  |

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

|  |  |
| --- | --- |
| Algorithm BFS(G)  initResult(G)  for all u ∈ G.vertics() do  setLabel(u, UNEXPECTED)  postInitVertex(u)  for all e ∈ G.edges() do  setLabel(e, UNEXPECTED)  postInitEdge(e)  for all v ∈ G.vertices() do  if isNextComponent(G, v)  preComponentVisit(G, v)  BFScomponent(G, v)  postComponentVisit(G, v)  return result(G)  Algorithm isNextComponent(G, v)  Return getlabel(v) = UNEXPLORED | Algorithm BFScomponent(G, s) {1 component}  setLabel(s, VISITED)  Q := new empty Queue  Q.enqueue(s)  startBFScomponent(G,s)  while ! Q.isEmpty() do  v := Q.dequeue()  preVertexVisit(G, v)  for all e ∈ G.incidentEdges(v) do  preEdgeVisit(G, v, e)  if getLabel(e) = UNEXPLORED  w := opposite(v, e)  edgeVisit(G, v, e, w)  if getLabel(w) = UNEXPLORED  preDiscEdgeVisit(G, v, e, w)  setLabel(e, DISCOVERY)  setLabel(w, VISITED)  Q.enquue(w)  postDiscEdgeVisit(G, v, e, w)  else  setLabel(e, CROSS)  crossEdgeVisit(G, v, e, w)  postVertexVisit(G, v)  finishBFScomponent(G, s) |

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 findPath(G, u, v)  return BFS(G, v)  Algorithm initResult(G)  minPath := new Sequence  visitedV := new Dictionary  isFinish = False    Algorithm preDiscEdgeVisit(G, v, e, w)  if !isFinish then  if !visitedV.haskey(v) then  minPath.insertLast(w)  visitedV.insertItem(v, 1)  if w = u then  isFinish = True  else  visitedV.insertItem(w, 1)    Algorithm result(G)  return minPath |  |

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 findCycle(G)  return BFS(G)  Algorithm initResult(G)  result := new Sequence  Algorithm crossEdgeVisit(G, v, e, w)  S1 = new Stack  createNewPath(v, S1)  S2 = new Stack  createNewPath(w, S2)  C = createNewPathCycle(S1, S2, e)  result.insertLast(C)  Algorithm result(G)  return result |  |

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, the template version of DFS is not meant to find the path between two vertices with the minimum number of edges. It is like this blow example.**

**A picture containing diagram, line, plot

Description automatically generated**

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.