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 initResult(G)

R<-new Sequence

Algorithm preComponentVisit(G, v)

S<-new Sequence

Algorithm postComponentVisit(G, v)

R.insertLast(S)

Algorithm startVertexVisit(G, v)

S.insertLast(v)

preDiscoveryTraversal(G, v, e, w)

S. insertLast(e)

Algorithm result(G)

return R

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

|  |  |
| --- | --- |
| Algorithm BFS(G)  Iterator verticesIter <- G.vertices()  while verticesIter.hasNext() do  v <- verticesIter.nextItem()  setLabel(v, UNEXPLORED)    Iterator edgesIter <- G.edges()    while edgesIter.hasNext() do  e <- edgesIter.nextItem()  setLabel(e, UNEXPLORED)  Q <- new Queue()    v <- G.aVertex()  setLabel(v, VISITED)    Q.enqueue(v)    while not Q.isEmpty() do  v <- Q.dequeue()  for each e in G.incidentEdges(v) do  w <- G.opposite(v, e)  if getLabel(w) = UNEXPLORED then  setLabel(e, DISCOVERY)  setLabel(w, VISITED)  Q.enqueue(w)  else  setLabel(e, BACK) | //BSF applying the template method  Algorithm BFS\_TemplateMethod(G, startV)  initResult(G)  Iterator verticesIter <- G.vertices()  while verticesIter.hasNext() do  vt <- verticesIter.nextItem()  setLabel(vt, UNEXPLORED)  preInitVertex(vt)  Iterator edgesIter <- G.edges()  while edgesIter.hasNext() do  e <- edgesIter.nextItem()  setLabel(e, UNEXPLORED)  preInitEdge(e)  Q <- new Queue()    //v <- G.aVertex()  setLabel(v, VISITED)  startVertexVisit(G, v)    Q.enqueue(startV)  while not Q.isEmpty() do  v <- Q.dequeue()  for each e in G.incidentEdges(v) do  w <- G.opposite(v, e)  if getLabel(w) = UNEXPLORED then  preDiscoveryTraversal(G, v, e, w)  setLabel(e, DISCOVERY)  setLabel(w, VISITED)  Q.enqueue(w)  postDiscoveryTraversal(G, v, e, w)  else  setLabel(e, BACK)  backTraversal(G, v, e, w)    finishVertexVisit(G, startV) //Template |

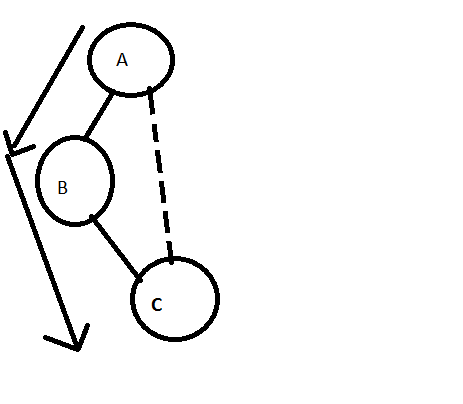
b. Write a pseudo code function findPath(G, u, v) that uses your Template Method from (a) to find a path in G between vertices u and v with the minimum number of edges, or report that no such path exists. Hint: Override the appropriate methods so that given two vertices u and v of G, your call to BFS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

|  |
| --- |
| Algorithm findPath(G, u, v)  S <- new Stack  minPath <- 0  BFS\_TemplateMethod(G, v)  return minPath  initResult(G)  minVertex <- v  minEdges <- 0    startVertexVisit(G, v)  setParent(v, 0)  setLevel(v, 0)  postDiscoveryTraversal(G, v, e, w)  setParent(w, e)  l <- getLevel(v) + 1  setLevel(w, l)  if w = u /\ l < minEdges then  minVertex <- w  minEdges = l    finishVertexVisit(G, v)  if minVertex = v then  return minPath    //Using the backtracking to find the path with minimum number of edges  S <- new Stack  z <- minVertex  while z <> v do  S.push(z)  e <- z.getParent()  S.push(e)  z <- G.opposite(z, e)    S.push(v)  minPath <- S.elements()  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.

|  |
| --- |
| findCycle(G)  cycle <- new Sequence  startV <- G.aVertex()  BFS\_TemplateMethod(G, startV)    return cycle    startVertexVisit(G, v)  setParent(v, 0)    backTraversal(G, v, e, w)  // How to trace back the cycle???  D1 <- new Dictionary(HT)  D1.insertItem(v, v.getParent())  D2 <- new Dictionary(HT)  D2.insertItem(w, w.getParent())  f <- false  while f = false do  v <- G.opposite(v, v.getParent())  w <- G.opposite(w, w.getParent())  if(v = w)  f <- true  else  p <- D2.findElement(v)  if p <> NO\_SUCH\_KEY then  makeCycle(cycle, D2, p, D1)  f <- true  else  D1.insertItem(v, v.getParent())    p <- D1.findElement(w)  if p <> NO\_SUCH\_KEY then  makeCycle(cycle, D1, p, D2)  f <- true  else  D2.insertItem(w, w.getParent())  makeCycle(cycle, B1, p, B2)  //Output p items from D2  Vs <- B1.keys()  Es <- B1.elements()  i <- 0  for each (k,f) in B1.items() do  cycle.insertLast(k)  cycle.insertLast(f)  i++  if i = p then //just get p items  break  //Output edge e between v & w  cycle.insertFirst(e)    //Output all items from D1  for each (k,f) in B2.items() do  cycle.insertFirst(k)  cycle.insertFirst(f) |

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.



If we see above example, where DFS is used to traverse a graph, To find path from node A to C,

It will traverse from node A to B, then from B to C. Even though shortest path is from A to C. So it’s not guaranteed that template version or non-template version of DFS algorithm will find minimum edges path between two vertices.

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.