Name: Walid Sultan Aly Ahmed

Assignment 13

1) **Algorithm** initResult(G)

Input: graph G
Output: create new empty sequence
S ← create new sequence

Algorithm preComponentVisit(G, v)

Input: graph G and vertex v
Output: add vertex v to a global sequence
s.insertLast(v)

Algorithm result(G)

Input: graph G
Output: sequence S containing a vertex from each connected component
Return s

2) a)

Algorithm BFS(G)

Input graph G
Output labeling of the edges and partition of the vertices of G
initResult(G)
for all u ← G.vertices()
setLabel(u, UNEXPLORED)
for all e ← G.edges()
setLabel(e, UNEXPLORED)
for all v ← G.vertices()
if getLabel(v) ←UNEXPLORED
preComponentVisit(G, v)
BFS(G, v)
postComponentVisit(G, v)
result(G)

Algorithm BFS(G, s)

L_0 ← new empty sequence
L_0.insertLast(s)
setLabel(s, VISITED)
i ← 0
while not Li.isEmpty()
L_{i+1} ←new empty sequence
for all v ∈ L_i.elements()
preVertexVisit(G, v)

```
                for all e Є G.incidentEdges(v)
                    preEdgeVisit(G, v, e)
                    if getLabel(e) = UNEXPLORED
                        w ← opposite(v,e)
                        if getLabel(w) = UNEXPLORED
                            preDiscoveryEdgeVisit (G, v, e, w)
                            setLabel(e, DISCOVERY)
                            setLabel(w, VISITED)
                            L_{i+1}.insertLast(w)
                            postDiscoveryEdgeVisit (G, v, e, w)
                        else
                            setLabel(e, CROSS)
                            crossEdgeVisit(G, v, e, w)
                    postEdgeVisit(G, v, e)
                postVertexVisit(G, v)
            i ← i+1
```

b)

```
Algorithm initResult(G)
    S ← create new stack

Algorithm preDiscoveryEdgeVisit(G, v, e, w)
    w.setParent(v)
    w.setEdge(e)
    if w = z then
        tracePath(w)

Algorithm tracePath(v)
    s.push(v)
    if v.parent() ¬= null then
        s.push(v.edge())
        tracePath(v.getParent())

Algorithm result(G)
    return s.elements()
```

c)

```
Algorithm initResult(G)
        S ← create new stack

Algorithm preDiscoveryEdgeVisit(G, v, e, w)
        If v.getLevel() = null then
                currentLevel ← 0
        else
                currentLevel ← v.getLevel()
        w.setLevel (currentLevel)
        w.setParent(v)
        w.setEdge(e)

Algorithm crossEdgeVisit(G, v, e, w)
        j ← v.getLevel()
        k ← w.getLevel()
        If j > k then
                While j>k do
                        s.insertLast(v)
                        s.insertLast(v.getEdge())
                        v ← v.getParent()
                        j ← j-1
        else if k > j then
                while k>j do
                        S.insertLast(w)
                        S.insertLast(w.getEdge())
                        w ← w.getParent()
                        k ← k-1
        t ← create new stack
        while v.getParent()¬= w.getParent() do
                S.insertLast(v)
                S.insertLast(v.getEdge())
                v ← v.getParent()
                t.push(w)
                t.push(w.getEdge)
        while ¬ t.isEmpty()  do
                S.insertLast(t.pop())

Algorithm result(G)
        return S.elements()
```

d) No, because DFS goes to the deepest level before it backtracks. As a result, It won't necessarily find the shortest path.

3)

```
Algorithm DijkstraDistances(G, s)
        Q ← new heap-based priority queue
        initResult(G, s)
        for all v in G.vertices()
                if v = s then
                        setDistance(v, 0)
                else
                        setDistance(v, inf)
                l ← Q.insert(getDistance(v), v)
                setLocator(v,l)
        while not Q.isEmpty()
                u ← Q.removeMin()
                for all e in G.incidentEdges(u)
                        { relax edge e }
                        z ← G.opposite(u,e)
                        preEdgeRelax(G, u, z, e)
                        r ← getDistance(u) + weight(e)
                        if r < getDistance(z)
                                preDistanceUpdate(G,u,z,e,r)
                                setDistance (z,r)
                                Q.replaceKey(getLocator(z),r)
                                postDistanceUpdate(G,u,z,e,r)
                        postEdgeRelax(G, u, z, e)
        Result(G,s)

Algorithm shortestPath(G, u, v)
        Input: graph G, vertex u and vertex v
        Output: shortest path between u and v
        target ← v
        DijkstraDistances (G, u)

Algorithm preDistanceUpdate(G,u,z,e,r)
        z.setEdge(e)

Algorithm Result(G,s)
        T ← create new sequence
        While target.getEdge() ¬= null do
                T.insertLast(target)
                T.insertLast(target.getEdge())
                target ← G.getOpposite(target, target.getEdge())

        result ← T.getElements()
```

4)

Algorithm initResult(G)

componentIndex ← 0

Algorithm preComponentVisit(G,v)

componentIndex ← componentIndex +1

Algorithm startVertexVisit(v)

v.setLabel(componentIndex)