procedure sortPipes(pipes[0 to n]: Pipe)

for i := 1 to n

minIndex := i

for j := 1 to n

if pipe[j].getDistance() < pipe[minIndex].getDistance() then minIndex := j

temp := pipes[minIndex]

pipes[minIndex] := pipes[i]

pipes[i] := temp

OU

swap pipes[minIndex] and pipes[i]

procedure find(parent[0 to n]: Integer, i:Integer)

while parent[i] ≠ i

i := parent[i)

return i

procedure union(parent[0 to n]:Integer, rank[0 to n]:Integer, x[0 to n]:Integer, y[0 to n]: Integer)

xRoot := find(parent, x)

yRoot := find(parent, y)

if rank[xRoot] < rank[yRoot] then parent[xRoot] := yRoot

else if rank[xRoot] > rank[yRoot] then parent[yRoot] := xRoot

else parent[yRoot] := xRoot then increment rank[xRoot]

procedure isDuplicate(vertices[0 to n ]: text, designation; text)

for each vertex in vertices

if vertex = designation then return false

return true

procedure findNumVertices(pipes[0 to n]: Pipe)

vertices = []

for each pipe in pipes

designationX := pipe.getWaterPoint\_X().getDesignation()

designationY := pipe.getWaterPoint\_Y().getDesignation()

if Call isDuplicate with vertices and designationX then

Add designationX to vertices

If Call isDuplicate with vertices and designationY then

Add designationY to vertices

return vertices

procedure kruskalMinSpanningTree(pipes[0 to n]:Pipe)

Call sortPipes with pipes

vertices[0 to n] = Call findNumVertices with pipes

verticeMax := n

minSpanningTree[0 to n]

parent[] := Integer[verticeMax]

rank[] := Integer[verticeMax]

for i := 0 to verticeMax

parent[i] := i

rank[i]:= 0

edgesAdded := 0

pipeIndex := 0

while edgesAdded < verticeMax – 1 and pipeIndex < n

currentPipe := pipes[pipeIndex]

x:= index of currentPipe.getWaterPoint\_X().getDesignation() IN vertices

y:= index of currentPipe.getWaterPoint\_Y().getDesignation() IN vertices

xRoot := Call find with parent and x

yRoot = Call find with parent and y

if xRoot ≠ yRoot then

add currentPipe into minSpanningTree

edgesAdded := edgesAdded + 1

Call union with parent, rank, xRoot, yRoot

pipeIndex := pipeIndex + 1

return minSpanningTree

Procedure findShortestPath(source: SignalPoint, target:SignalPoint, signalPoints[0 to n]:SignalPoint, routes[0 to n]: Route)

numPoints := n (size of signalPoints)

distances[] := Integer[numPoints]

visited[] := Boolean[numPoints]

previous[] := Integer[numPoints]

for i := 0 to numPoints

distances[i] := Maximum value of an Integer

visited[i] := false

previous[i] := -1

sourceIndex := signalPoints.indexOf(source)

distances[sourceIndex] := 0

for i := 0 to numPoint – 1

closest := -1

closestDistance = Maximum value of an Integer

for j := 0 to numPoints

if visited[j] := false and distances [j] < closestDistance then

closest := j

closestDistance := distance[j]

if closest = -1 then

leaves for

visited[closest] := true

for each route in routes

fromIndex := signalPoints.indexOf(route.getS1())

toIndex := signalPoints.indexOf(route.getS2())

if fromIndex = closest and visited[toIndex] 0= false then

newDist = distances[closest] + total distance of route

if newDist < distances[toIndex] then

distances[toIndex] = newDist

previous[toIndex] = closest

if toIndex = closest and visited[fromIndex] = false then

newDist = distances[closest] + total distance of route

if newDist < distances[fromIndex] then

distances[fromIndex] = newDist

previous[fromIndex] = closest

path[0 to n]

for at = signalPoints.indexOf(target) to at ≠ -1

add vertex to the begining of the path

if path = null or path.get(0) ≠ source then

return new array

return Call constructRoute with path and routes

procedure constructRoute (signalPoints[0 to n]:SignalPoint, routes[0 to n]:Route)

newRoute[0 to n]

for i := 0 to n (size of signalPoints)

for each route in routes

if route.equals(new Route(signalPoints.get(i), signalPoints.get(i + 1))then

newRoute.add(new Route(route.getDistance(), signalPoints.get(i), signalPoints.get(i + 1)));

return newRoute

procedure findShortestPathToNearestAP(source:SignalPoint, signalPoints[0 to n]:SignalPoint, routes[0 to n]:Route, isAssemblyPoint:Boolean)

numPoints = n (size of signalPoints)

distances[] := Integer[numPoints]

visited[] := Boolean[numPoints]

previous[] = Integer[numPoints]

for i:= 0 to numPoints

distances[i] := Maximum value of Integer

visited[i] := false

previous[i] := -1

sourceIndex := signalPoints.indexOf(source);

distances[sourceIndex] := 0

for i := 0 to numPoints -1

closest := -1

closestDistance := Maximum value of an Integer

for j:= 0 to numPoints

if visited[j] = false and distances[j] < closestDistance then

closest := j

closestDistance = distances[j]

if closest = -1 then

leaves for

visited[closest] := true

for each route in routes

fromIndex := signalPoints.indexOf(route.getS1());

toIndex := signalPoints.indexOf(route.getS2());

if fromIndex = closest and visited[toIndex] := false then

newDist := distances[closest] + total distance of route

if newDist < distances[toIndex] then

distances[toIndex] := newDist

previous[toIndex] := closest

path[0 to n]

nearestAPIndex := Call findNearestAssemblyPointIndex with signalPoints, distances, isAssemblyPoint

for at :=nearestAPIndex to at ≠ -1

add vertex to the begining of the path

if path = null or first vertex of path ≠ source then

return new array

return Call constructRoute with path and routes