```python
from epyt import epanet
import numpy as np
import os

# Load EPANET network
inp_file = 'singlesourcenetwork.inp'  # Path to your input file
d = epanet(inp_file)  # Load the network file

def run_DDA():
    """Run the hydraulic analysis for the current state of the network."""
    d.openHydraulicAnalysis()
    d.initializeHydraulicAnalysis()
    tstep = 1
    while tstep > 0:
        d.runHydraulicAnalysis()
        tstep = d.nextHydraulicAnalysisStep()
    d.closeHydraulicAnalysis()

    NodeCount=d.getNodeCount()
    NodeID=d.getNodeJunctionNameID()
    JunctionCount = d.getNodeJunctionCount()
    PipeCount=d.getLinkCount()

    print(f"\n The network consisits of {NodeCount} nodes of which
{JunctionCount} are Junction Nodes having Node ID {NodeID} along with
{PipeCount} pipes.")

    print (f"\n The Initial Pressure for all {NodeCount} nodes are as
follows:")

    for i in range (1, JunctionCount +1):
        print(f"Node {i+1}: {d.getNodePressure(i):.2f} m")

    print (f"\n The Initial Status for all {PipeCount} pipes are as follows:")

    for i in range (1, PipeCount +1):
        print(f" Pipe {i}: {d.getLinkStatus(i)}: {d.getLinkFlows(i):.2f} lps")

def get_Pressure(NodeCount):
    Pressures = []
    for i in range(NodeCount):
        try:
```

```
            pressure = d.getNodePressure(i)
            Pressures.append(pressure)

        except Exception as e:
            print(f"Error retrieving pressure for node {i}: {e}")
            Pressures.append(None)  # Handle this as needed
    return Pressures


def isolatePipe(PipeIsolate, NodeCount):
    try:
        PipeName = d.getLinkPipeNameID(PipeIsolate)
        d.getLinkStatus(PipeName)
        status = 0
        d.setLinkStatus(PipeName, status)
        current_status = d.getLinkStatus(PipeIsolate)


        print(f"\n Pipe {PipeIsolate} is successfully isolated with Current
Status as {current_status}: Closed")
    except Exception as e:
        print(f"Error isolating pipe {PipeIsolate}: {e}")

    # Debug information for node IDs
    node_links = d.getNodesConnectingLinksID(PipeName)
    print(f"\nNodes connected to pipe {PipeName}: {node_links}")

    # Iterate over the node IDs and convert them to integers for validation
    if len(node_links) == 0:
        print(f"Error: No nodes found connected to pipe {PipeIsolate}.")
        return [None]*NodeCount

    for node in node_links[0]:  # Access the first element since node_links is
a nested array
        node_int = int(node)  # Convert node ID from string to integer
        if node_int < 0 or node_int >= NodeCount:
            print(f"Error: Node {node_int} is out of bounds.")
        else:
            print(f"Node {node_int} is valid.")

    d.openHydraulicAnalysis()
    d.runHydraulicAnalysis()
```

```
    Pressures = get_Pressure(NodeCount)

    print(f"\n The Nodes having Negative Pressures after Isolating Pipe
{PipeName} are as follows: ")

    for idx in range(NodeCount):  # Use range to ensure we check all nodes up
to NodeCount
        try:
            pressure = Pressures[idx]  # Access the pressure for the current
node
        # Check if the pressure is defined and less than zero
            if pressure is not None and pressure < 0:
                 print(f" Node {idx + 1} has a negative pressure:
{pressure:.2f}")
        except IndexError:
             print(f"Error: Node index {idx + 1} is out of bounds.")  # Should
not occur, but added for safety
        except Exception as e:
             print(f"An unexpected error occurred while checking pressure for
Node {idx + 1}: {e}")

    return Pressures

def get_critical_node(Pressures, NodeCount):
    critical_node = None
    max_pressure_deficiency = float('inf')
    critical_node_coordinates = None

    for i in range(1, NodeCount + 1):
        Pressure_after_modification = Pressures[i - 1]

               # Check for maximum negative pressure
        if Pressure_after_modification < 0 and Pressure_after_modification <
max_pressure_deficiency:
            max_pressure_deficiency = Pressure_after_modification
            critical_node = i

    if critical_node is not None:
        NodeCoords = d.getNodeCoordinates()
        critical_node_index = critical_node - 1
```

```
        if critical_node_index >= 0 and critical_node_index <
len(NodeCoords['x']):
            x_coord = NodeCoords['x'][critical_node_index]
            y_coord = NodeCoords['y'][critical_node_index]
            critical_node_coordinates = (x_coord, y_coord)
        else:
            print(f"Error: No coordinates found for critical node
{critical_node}.")

            critical_node_coordinates = None  # Set to None if index is
invalid

        print(f"The Critical node is identified as Node ID {critical_node}
having coordinates {critical_node_coordinates} with maximum negative pressure:
{max_pressure_deficiency:.2f}")
    else:
        print("No critical node with negative pressure found.")

    return critical_node, critical_node_coordinates, max_pressure_deficiency


def check_critical_node_pressure(critical_node, max_pressure_deficiency):
    HGL_Source = d.getNodeElevations(d.getNodeReservoirIndex())[0]

    HGL_critical_node=d.getNodeElevations(critical_node-1)

    if isinstance(HGL_critical_node, (list, np.ndarray)):
        HGL_critical_node = HGL_critical_node[0]  # Assuming you want the
first element

    print(f"\n HGL Source: {HGL_Source}, HGL min at Critical Node
{critical_node}: {HGL_critical_node}")

    HGL_difference = HGL_critical_node - HGL_Source
    print(f"The HGL difference is :{HGL_difference:.2f}")

    should_attach = max_pressure_deficiency > HGL_difference
    if should_attach:
        print(f"\n Max pressure deficiency of {max_pressure_deficiency:.2f} at
critical node {critical_node} "
```

```
            f"is less than the HGL difference {HGL_difference:.2f}.
Attaching CR to critical node {critical_node}.")
    else:
        print(f"\n Max pressure deficiency of {max_pressure_deficiency:.2f} at
critical node {critical_node} "
            f"is greater than the HGL difference {HGL_difference:.2f}. Set
demand at critical node {critical_node} as zero.")

    return should_attach, HGL_critical_node

def set_demand_zero_and_run(critical_node, NodeCount):
    # Retrieve the actual name or index of the critical node
    nameID = d.getNodeNameID(critical_node)
    index=d.getNodeIndex(nameID)
    print(f"The index of the critical node is {index}")

    # Retrieve the base demands for all nodes
    basedemand = d.getNodeBaseDemands()
    demand_array = basedemand[1]  # Access the array of base demands

    print("Nodes with their base demands before adjustments:")
    for i in range(1,NodeCount):
    # Access the demand for each node by indexing into demand_array
        base_demand_value = demand_array[i-1]  # Get demand for node i
        print(f"Node {i+1}: {base_demand_value:.2f} lps")

    demand_critical_node=d.getNodeBaseDemands()[1][index]

    # Check if the critical node matches and set its demand to zero
    print(f"The base demand at critical node {critical_node} is
{demand_critical_node} lps")
    demand = 0
    # demand_array[critical_node - 1] = 0  # Set the demand to zero for the
critical node
    d.setNodeBaseDemands(index-1, demand)  # Update the demand in the model
    print(f"Demand at critical node {critical_node} has been set to zero.")

    d.openHydraulicAnalysis()
    d.runHydraulicAnalysis()
    d.closeHydraulicAnalysis()

    updated_basedemand=d.getNodeBaseDemands()
```

```
       # Retrieve and print the updated base demands after running analysis
    updated_demand_array = updated_basedemand[1]
    print("Nodes with their base demands after adjustments:")

    for i in range(1,NodeCount):
        updated_base_demand_value = updated_demand_array[i-1]
        print(f"Node {i + 1}: Base Demand = {updated_base_demand_value:.2f}
lps")

    # # Retrieve the updated pressures after the hydraulic analysis

    Pressures = get_Pressure(NodeCount)

    print(f"The Nodes having Negative Pressures after setting demand as zero
to critical Node {critical_node} are as follows:")

# Initialize an empty list to store nodes with negative pressures
    nodes_with_negative_pressures = []

    for idx in range(NodeCount):  # Use range to ensure we check all nodes up
to NodeCount
        try:
        # Skip the critical node (Node 3 in this case)
            if idx + 1 == critical_node:
                continue  # Skip this iteration for the critical node

            pressure = Pressures[idx]  # Access the pressure for the current
node

        # Check if the pressure is defined and less than zero
            if pressure is not None and pressure < 0:
               print(f"Node {idx + 1} has a negative pressure:
{pressure:.2f}")
               nodes_with_negative_pressures.append((idx + 1, pressure))  #
Append the node ID and its pressure
        except IndexError:
            print(f"Error: Node index {idx + 1} is out of bounds.")  # Should
not occur, but added for safety

        except Exception as e:
            print(f"An unexpected error occurred while checking pressure for
Node {idx + 1}: {e}")
```

```
# Find the node with the most negative pressure (i.e., the minimum pressure)
    if nodes_with_negative_pressures:
        next_critical_node = min(nodes_with_negative_pressures, key=lambda x:
x[1])  # Min pressure
        next_node_id, next_node_pressure = next_critical_node
        NodeCoords = d.getNodeCoordinates()
        next_node_index = next_node_id - 1  # Adjust for zero-indexing
        if next_node_index >= 0 and next_node_index < len(NodeCoords['x']):
            x_coord = NodeCoords['x'][next_node_index]
            y_coord = NodeCoords['y'][next_node_index]
            next_node_coordinates = (x_coord, y_coord)
            print(f"\nNode ID {next_node_id} is the next Critical Node with a
Negative Pressure: {next_node_pressure:.2f} with Coordinates: ({x_coord},
{y_coord})")
        else:
            print(f"Error: No coordinates found for Node {next_node_id}")
            next_node_coordinates = None

        return next_node_id, next_node_pressure, next_node_coordinates
    else:
        print("\nNo other nodes have negative pressures.")
        return None, None, None

def delete_complementary_reservoir(reservoirID, PipeID):

    try:
        # Delete the pipe connected to the reservoir first
        if d.getLinkIndex(PipeID) is not None:
            d.deleteLink(PipeID)
            print(f"Deleted pipe with ID {PipeID}.")


        # Delete the reservoir node
        if d.getNodeIndex(reservoirID) is not None:
            d.deleteNode(reservoirID)
            print(f"Deleted reservoir with ID {reservoirID}.")


    except Exception as e:
        print(f"Error occurred while deleting the complementary reservoir or
pipe: {e}")
```

```
def attach_complementary_reservoir_with_pipe(critical_node,
HGL_critical_node,critical_node_coordinates):
    d.closeHydraulicAnalysis()
    reservoirID = f"CR_{critical_node}"
    print(f"Attaching complementary reservoir to Node {critical_node} with
elevation of {HGL_critical_node}.")

    # Get node coordinates and determine reservoir position
    NodeCoords = critical_node_coordinates
    print(f"The coordinates for critical node {critical_node} are :
{NodeCoords}")

    x_coord = NodeCoords[0] + 50
    y_coord = NodeCoords[1] + 50
    CR_coords = [x_coord, y_coord]

    print(f"Reservoir will be placed at coordinates with offset: {CR_coords}
at elevation {HGL_critical_node}")

    # Add reservoir to the model
    res_index = d.addNodeReservoir(reservoirID)

    if res_index < 0:
        print(f"Error:Failed to add reservoir {reservoirID}")
        return None, None

    d.setNodeElevations(res_index, HGL_critical_node)
    d.setNodeCoordinates(res_index, CR_coords)
    print(f"Reservoir {reservoirID} successfully added with index
{res_index}.")
    d.plot()

    initial_pipe_count = len(d.getLinkNameID())

    pipeID = f"{initial_pipe_count + 1}"  # Sequential pipe ID
    fromNode = reservoirID
    toNode = str(critical_node)

    length = 0.1
    diameter = 350
    roughness = 130
```

```
    # Add the pipe
    PipeIndex = d.addLinkPipe(pipeID,fromNode, toNode,length, diameter,
roughness)
    d.getLinkPipeCount()
    d.getLinkIndex(pipeID)
    PipeLength=d.getLinkLength(PipeIndex)
    PipeDia=d.getLinkDiameter(PipeIndex)
    PipeRw=d.getLinkRoughnessCoeff(PipeIndex)

    if PipeIndex >= 0:
        print(f"Pipe {pipeID} was successfully added with length
{PipeLength:.2f} m, dia {PipeDia}mm and roughness {PipeRw}.")
    else:
        print(f"Failed to add pipe {pipeID}.")
        return res_index, None

    nodes_connected_to_pipe = d.getNodesConnectingLinksID(pipeID)
    print(f"Nodes connected to pipe {pipeID} : {nodes_connected_to_pipe}")

    # Check if the pipe exists in the model
    all_link_ids = d.getLinkNameID()
    if pipeID in all_link_ids:
        print(f"Pipe {pipeID} is present in the model with ID {PipeIndex}.")
    else:
        print(f"Pipe {pipeID} is not recognized in the model.")

        return res_index,PipeIndex
    d.plot()
    d.saveInputFile()

def
check_supply_and_demand(critical_node,reservoirID,pipeID,HGL_critical_node):
    """Check if the CR's inflow meets the demand at the critical node."""
    inflow_value = 0

        # Get the link flows (inflow from the complementary reservoir)
    try:
        inflow = d.getLinkFlows(d.getLinkIndex(pipeID))

        if isinstance(inflow, dict):
```

```
            inflow_value = inflow.get('flow', 0)  # Replace 'flow' with the
appropriate key if necessary
        else:
            inflow_value = inflow


        print(f"The Supply from CR through Pipe {pipeID} is:
{inflow_value:.2f} lps to meet shortage.")

        # print(f"Base demand structure at {critical_node} is: {base_demand}")

        base_demand = d.getNodeBaseDemands(critical_node)
        base_demand_value = list(base_demand.values())[0][0]
        print(f"Base demand at node {critical_node} is:
{base_demand_value:.2f} lps")


        actual_supply = base_demand_value - inflow_value  # Compare inflow to
base demand
        print(f"Hence, Actual supply of {actual_supply:.2f} is possible from
critical node {critical_node} with required HGL of the critical node as
{HGL_critical_node:.2f} m")


        if actual_supply <= base_demand_value:
            print(f"Flow from CRS is shortage to critical node
{critical_node}.")
        else:
            print(f"Supply from CR exceeds demand at critical node
{critical_node}. Setting demands as zero to critical node {critical_node} and
disconnect CR. And Simulate the Network.")

            # d.setNodeBaseDemands(critical_node, 0)  # Set demand to zero

            # d.deleteNode(reservoirID)  # Cleanup
            # d.deleteLink(pipeID)

            # print(f"The Actual supply {actual_supply} exceeded demand at CN
{critical_node}. Deleting complementary reservoir.")

        # Re-run hydraulic analysis to reflect changes
        return inflow_value
```

```
    except Exception as e:
        print(f"An error occurred while checking supply and demand at node
{critical_node}: {e}")

    return inflow_value


def check_negative_pressures_after_cr(Pressure_after_CR, critical_node,
inflow_value,excluded_nodes):

    for i, pressure_at_node in enumerate(Pressure_after_CR[1:], start=2):
        if i in excluded_nodes:
            continue
        if i == critical_node:  # Handle the critical node separately
            print(f"Node {i} was the critical node. Pressure is allowed to be
negative here: {pressure_at_node:.2f}.")
        elif i in excluded_nodes:  # Check if demand was set to zero at the
node
            print(f"Node {i} is the next critical node with a negative
pressure of {pressure_at_node:.2f}, as demand at Node {i} is set as zero.")
        else:
            if pressure_at_node < 0:  # Check if the pressure is negative
                print(f"Node {i} has a negative pressure of
{pressure_at_node:.2f}.")
            else:
                print(f"Node {i} has positive pressure:
{pressure_at_node:.2f}.")


def run_simulation(HGL_Source, PipeIsolate):
# Step 1: Run initial hydraulic analysis
    print("Now, Running Hydraulic Analysis.")
    run_DDA()
    print(" After Initial hydraulic Analysis. Now next STEP 1 - ISOLATE THE
PIPE ")

    NodeCount = d.getNodeCount()
    PipeCount=d.getLinkCount()

# Step 2 Isolate the specified pipe and get updated pressures
    isolatePipe(PipeIsolate, NodeCount)
```

```
    print(f"Pipe {PipeIsolate} status after simulation:
{d.getLinkStatus(PipeIsolate)}")

    iteration_count = 0
    max_iterations = 100 # Set a maximum iteration count to prevent infinite
loops
    convergence_threshold = 0.1  # Relaxed convergence threshold
    previous_max_deficiency = None
    inflow_value=0
    excluded_nodes= []


    while iteration_count < max_iterations:

# Step 3 Find the critical node and its pressure deficiency
        print(f"Iteration {iteration_count + 1}...")

        Pressures = get_Pressure(NodeCount)  # Get the updated pressures

        critical_node, critical_node_coordinates, max_pressure_deficiency =
get_critical_node(Pressures, NodeCount)

        excluded_nodes.append(critical_node)

        print(f"The excluded nodes: {excluded_nodes}")

        # Check critical node pressure and pass HGL_critical_node
        print(f"\n Next step is to check the Pressure at the Critical Node
identified....")

        should_attach, HGL_critical_node =
check_critical_node_pressure(critical_node, max_pressure_deficiency)

        if should_attach:

attach_complementary_reservoir_with_pipe(critical_node,HGL_critical_node,criti
cal_node_coordinates)  # Pass HGL_critical_node here
            reservoirID = "CR_{}".format(critical_node)
            initial_pipe_count = len(d.getLinkNameID())
            pipeID = f"{initial_pipe_count}"

            d.openHydraulicAnalysis()
```

```
            d.runHydraulicAnalysis()
            d.closeHydraulicAnalysis()


inflow_value=check_supply_and_demand(critical_node,reservoirID,pipeID,HGL_crit
ical_node)

            Pressure_after_CR=get_Pressure(NodeCount)

            print("The Pressure at nodes after attaching CRS are as follows:")

            for i, pressure_at_node in enumerate(Pressure_after_CR[1:],
start=2):
                if i in excluded_nodes:
                    continue
            if i == critical_node:  # Handle the critical node separately
                print(f"Node {i} was the critical node. Pressure is allowed to
be negative here: {pressure_at_node:.2f}.")
            elif i in excluded_nodes:  # Check if demand was set to zero at
the node
                print(f"Node {i} is the next critical node with a negative
pressure of {pressure_at_node:.2f}, as demand at Node {i} is set as zero.")
            else:
                if pressure_at_node < 0:  # Check if the pressure is negative
                    print(f"Node {i} has a negative pressure of
{pressure_at_node:.2f}.")
                else:
                    print(f"Node {i} has positive pressure:
{pressure_at_node:.2f}.")

            all_positive = True

            if all_positive:

               print(f"Attaching CR to node {critical_node} has brought
positive pressure at all nodes, while CR supplies {inflow_value:.2f} lps to
meet shortage.")
            else:

                print(f"\nStarting Iteration {iteration_count + 1}...")

            if critical_node is None:
```

```
                    print(f"\nThe next critical node to add a CR is Node
{critical_node} with the maximum negative pressure:
{max_negative_pressure:.2f}.")
            else:
                print("No suitable nodes found for adding CR after excluding
the critical nodes.")
                basedemand=d.getNodeBaseDemands()
                demand_array = basedemand[1]  # Access the array of base
demands

                for i in range(1,NodeCount):
                    base_demand_value = demand_array[i-1]  # Get demand for
node i
                    print(f"Node {i+1}: {base_demand_value:.2f} lps:
{d.getNodePressure(i):.2f}")

                for i in range (1, PipeCount +1):
                    print(f" Pipe {i}: {d.getLinkStatus(i)}:
{d.getLinkFlows(i):.2f} lps")
        else:
            print(f"\nUnable to attach CR at iteration {iteration_count + 1}")

            print(f"\nStarting Iteration {iteration_count + 2}...")

            print(f"\nSetting demand to zero for Critical Node {critical_node}
and running analysis again.")

            next_node_id, next_node_pressure, next_node_coordinates =
set_demand_zero_and_run(critical_node, NodeCount)

            excluded_nodes.append(next_node_id)

            print(f"The excluded node : {excluded_nodes}")

            critical_node = next_node_id
            max_pressure_deficiency = next_node_pressure

            should_attach,
HGL_critical_node=check_critical_node_pressure(critical_node,max_pressure_defi
ciency)

            critical_node_coordinates = next_node_coordinates
```

```
            if should_attach:

attach_complementary_reservoir_with_pipe(critical_node,HGL_critical_node,criti
cal_node_coordinates)
                reservoirID = "CR_{}".format(critical_node)
                initial_pipe_count = len(d.getLinkNameID())
                pipeID = f"{initial_pipe_count}"

                d.openHydraulicAnalysis()
                d.runHydraulicAnalysis()
                d.closeHydraulicAnalysis()


inflow_value=check_supply_and_demand(critical_node,reservoirID,pipeID,HGL_crit
ical_node)

            print("\nCheck for Negative pressures in the network if any.....")

            Pressure_after_CR=get_Pressure(NodeCount)

            all_positive = True
            next_critical_node=None
            max_negative_pressure=float('inf')

            for i, pressure_at_node in enumerate(Pressure_after_CR[1:],
start=2):
                if i in excluded_nodes:
                     if i == critical_node:
                        print(f"Node {i} has negative pressure:
{pressure_at_node:.2f}, allowed to be negative as CR is attached.")
                     elif i == next_node_id:
                        print(f"Node {i} has negative pressure:
{pressure_at_node:.2f}, it is the next critical node.")
                else:
                    if pressure_at_node < 0:

                        all_positive = False
                        print(f"Node {i} has negative pressure:
{pressure_at_node:.2f}")
```

```
                    if i not in excluded_nodes and pressure_at_node <
max_negative_pressure:
                        max_negative_pressure = pressure_at_node
                        next_critical_node = i
                else:
                     print(f"Node {i} has positive pressure:
{pressure_at_node:.2f}")

            if all_positive:
               print(f"Attaching CR to node {next_node_id} has brought
positive pressure at all nodes, while CR supplies {inflow_value:.2f} lps to
meet shortage.")
            else:
                print(f"\nStarting Iteration {iteration_count + 3}...as
negative pressure is present...")

                if next_critical_node is not None:
                    print(f"\nThe next critical node to add a CR is Node
{next_critical_node} with the maximum negative pressure:
{max_negative_pressure:.2f}.")
                else:
                    print("No suitable nodes found for adding CR after
excluding the critical nodes.")

                    for i in range(1,NodeCount):
                        base_demand_value = d.getNodeActualDemand(i)  # Get
demand for node i

                    if i == next_node_id:
                        adjusted_supply=base_demand_value - inflow_value

                        print(f"Node {i}: Outflow = {adjusted_supply:.2f}
lps")
                    else:
                        print(f"Node {i+1}: Outflow = {base_demand_value:.2f}
lps")

                    for i in range (1, PipeCount +1):
                        print(f" Pipe {i}: {d.getLinkStatus(i)}:
{d.getLinkFlows(i):.2f} lps")

                    break
```

```
                excluded_nodes.append(next_critical_node)
                print(f"The excluded node : {excluded_nodes}")

            critical_node=next_critical_node
            max_pressure_deficiency=max_negative_pressure
            node_coordinates = d.getNodeCoordinates()

            if critical_node in node_coordinates['x']:
                   x_coord = node_coordinates['x'][critical_node-1]  # Get x
coordinate
                   y_coord = node_coordinates['y'][critical_node-1]  # Get y
coordinate

            HGL_critical_node=d.getNodeElevations(critical_node-1)
            print(f"The elevation is {HGL_critical_node}")
            critical_node_coordinates = (x_coord, y_coord)

            print(f"Coordinates for Critical Node {critical_node}:
{critical_node_coordinates}")

            if should_attach:
                    attach_complementary_reservoir_with_pipe(critical_node,
HGL_critical_node, critical_node_coordinates)
                    reservoirID = f"CR_{critical_node}"
                    initial_pipe_count = len(d.getLinkNameID())
                    pipeID = f"{initial_pipe_count}"
                    status=1
                    d.setLinkStatus(pipeID,status)
                    d.getLinkStatus(pipeID)

                    d.openHydraulicAnalysis()

                    d.initializeHydraulicAnalysis()


                    inflow = d.getLinkFlows(d.getLinkIndex(pipeID))
                    print (f"{inflow}")



            #     Pressure_after_CR=get_Pressure(NodeCount)
```

```
            #     all_positive = True
            #     next_critical_node=None
            #     max_negative_pressure=float('inf')

            #     for i, pressure_at_node in enumerate(Pressure_after_CR[1:],
start=2):
            #         if i in excluded_nodes:
            #             if i == critical_node:
            #                print(f"Node {i} has negative pressure:
{pressure_at_node:.2f}, allowed to be negative as CR is attached.")
            #             elif i == next_node_id:
            #                 print(f"Node {i} has negative pressure:
{pressure_at_node:.2f}, it is the next critical node.")
            #         else:
            #             if pressure_at_node < 0:
            #                 all_positive = False
            #                 print(f"Node {i} has negative pressure:
{pressure_at_node:.2f}")

            #             if i not in excluded_nodes and pressure_at_node <
max_negative_pressure:
            #                 max_negative_pressure = pressure_at_node
            #                 next_critical_node = i
            #             else:
            #                 print(f"Node {i} has positive pressure:
{pressure_at_node:.2f}")


            # if all_positive:

            #     print(f"Attaching CR to node {critical_node} has brought
positive pressure at all nodes, while CR supplies {inflow_value:.2f} lps to
meet shortage.")
            # else:

            #      print(f"\nStarting Iteration {iteration_count + 4}...as
negative pressure is present")

            # if next_critical_node is not None:
```

```
            #     print(f"\nThe next critical node to add a CR is Node
{next_critical_node} with the maximum negative pressure:
{max_negative_pressure:.2f}.")
            # else:
            #     print("No suitable nodes found for adding CR after excluding
the critical nodes.")

            #     for i in range(1,NodeCount):
            #         base_demand_value = d.getNodeActualDemand(i)  # Get
demand for node i

            #         if i == next_node_id and i == next_critical_node:
            #             adjusted_supply=base_demand_value - inflow_value

            #             print(f"Node {i}: Outflow = {adjusted_supply:.2f}
lps")
            #         else:
            #             print(f"Node {i+1}: Outflow =
{base_demand_value:.2f} lps")

            #         for i in range (1, PipeCount +1):
            #              print(f" Pipe {i}: {d.getLinkStatus(i)}:
{d.getLinkFlows(i):.2f} lps")
            #         break

            #     excluded_nodes.append(next_critical_node)
            #     print(f"The excluded node: {excluded_nodes}")

            #     critical_node=next_critical_node
            #     max_pressure_deficiency=max_negative_pressure

            #     node_coordinates = d.getNodeCoordinates()

            #     if critical_node in node_coordinates['x']:
            #         x_coord = node_coordinates['x'][critical_node-1]  # Get
x coordinate
            #         y_coord = node_coordinates['y'][critical_node-1]  # Get
y coordinate

            #         HGL_critical_node=d.getNodeElevations(critical_node-1)
            #         print(f"The elevation is {HGL_critical_node}")
            #         critical_node_coordinates = (x_coord, y_coord)
```

```
            #           # Check if the critical_node exists in the dictionary

            #         print(f"Coordinates for Critical Node {critical_node}:
{critical_node_coordinates}")

            #     if should_attach:
            #         attach_complementary_reservoir_with_pipe(critical_node,
HGL_critical_node, critical_node_coordinates)
            #         reservoirID = f"CR_{critical_node}"
            #         initial_pipe_count = len(d.getLinkNameID())
            #         pipeID = f"{initial_pipe_count}"
            #             #  d.setLinkStatus(pipeID,1)

            #         d.openHydraulicAnalysis()
            #         d.runHydraulicAnalysis()
            #         d.closeHydraulicAnalysis()

            #
inflow_value=check_supply_and_demand(critical_node,reservoirID,pipeID,HGL_crit
ical_node)

            #     print("\n Check for negative Pressure in the network if
any......")

            #     Pressure_after_CR=get_Pressure(NodeCount)

            #     all_positive = True
            #     next_critical_node=None
            #     max_negative_pressure=float('inf')

            #     for i, pressure_at_node in enumerate(Pressure_after_CR[1:],
start=2):
            #         if i in excluded_nodes:
            #             if i == next_node_id:
            #                     print(f"Node {i} has negative pressure:
{pressure_at_node:.2f}, allowed to be negative as CR is attached.")
            #             elif i == next_critical_node:
            #                     print(f"Node {i} has negative pressure:
{pressure_at_node:.2f}, it is the critical node.")
            #             else:
            #                 if pressure_at_node < 0:
            #                     all_positive = False
```

```
            #                     print(f"Node {i} has negative pressure:
{pressure_at_node:.2f}")
            #                     if i not in excluded_nodes and
pressure_at_node < max_negative_pressure:
            #                         max_negative_pressure = pressure_at_node
            #                         next_critical_node=i

            #                 else:
            #                     print(f"Node {i} has positive pressure:
{pressure_at_node:.2f}")

            #     if all_positive:

            #         print(f"Attaching CR to node {critical_node} has brought
positive pressure at all nodes.")
            #     else:

            #         print(f"\nStarting Iteration {iteration_count + 4}...")

            #     if next_critical_node is not None:
            #         print(f"\nThe next critical node to add a CR is Node
{next_critical_node} with the maximum negative pressure:
{max_negative_pressure:.2f}.")
            #     else:
            #         print("No suitable nodes found for adding CR after
excluding the critical nodes.")


            #         for i in range (1, PipeCount +1):
            #             print(f" Pipe {i}: {d.getLinkStatus(i)}:
{d.getLinkFlows(i):.2f} lps")

    #   After every simulation step or before running the hydraulic model:
        PipeIsolate = 3

# Ensure Pipe isolated remains closed
        if d.getLinkStatus(PipeIsolate)!= 0:
            d.setLinkStatus(d.getLinkPipeNameID(PipeIsolate), 0)  # Force the
pipe to stay closed

# Verify the status remains locked
```

```
            print(f"Locked Pipe {PipeIsolate} status:
{d.getLinkStatus(PipeIsolate)}")
 # After adding the reservoir and pipe:

        previous_max_deficiency = max_pressure_deficiency
        iteration_count += 1
        # Check for convergence
        if previous_max_deficiency is not None:
            pressure_difference = abs(previous_max_deficiency -
max_pressure_deficiency)
            print(f"Pressure difference from last iteration:
{pressure_difference:.3f}")

            if pressure_difference < convergence_threshold:
                print(f"Convergence achieved! Pressure difference is below
{convergence_threshold}. Stopping simulation.")
                break

        if iteration_count >= max_iterations:
            print("Reached maximum number of iterations without full
convergence. Exiting the loop.")
            break

    print("Simulation complete and network stabilized.")

HGL_Source=d.getNodeElevations(d.getNodeReservoirIndex())
PipeIsolate=3
run_simulation(HGL_Source,PipeIsolate)
```