```
import networkx as nx
import matplotlib.pyplot as plt
import sys
from collections import defaultdict

# Gabow's algorithm to find strongly
connected components
def gabow_scc(graph):
index = [0]
S = []
P = []
stack = []
lowlink = {}
scc_found = []
scc_result = []

def gabow(v):
lowlink[v] = index[0]
index[0] += 1
stack.append(v)
S.append(v)
```

```
                P.append(v)

                for w in graph[v]:
                if w not in lowlink:
                gabow(w)
                elif w not in scc_found:
                while lowlink[P[-1]] > lowlink[w]:
                P.pop()

                if P[-1] == v:
                scc = []
                while True:
                w = S.pop()
                scc_found.append(w)
                scc.append(w)
                if w == v:
                break
                scc_result.append(scc)
                P.pop()

                for v in graph:
```

```
                if v not in lowlink:
                    gabow(v)

        return scc_result

    # Jens Schmidt's algorithm to check for
      articulation points in undirected graph
    def jens_schmidt_ap(graph):
        def dfs(u, parent, discovery_time, low, ap,
                visited, time, child_count):
            visited[u] = True
            discovery_time[u] = low[u] = time[0]
            time[0] += 1
            children = 0

            for v in graph[u]:
                if not visited[v]:
                    parent[v] = u
                    children += 1
                    dfs(v, parent, discovery_time, low,
                        ap, visited, time, child_count)
```

```
                low[u] = min(low[u], low[v])
            if parent[u] is None and children > 1:
                ap[u] = True
            if parent[u] is not None and low[v]
            >= discovery_time[u]:
                ap[u] = True
            elif v != parent[u]:
            low[u] = min(low[u],
            discovery_time[v])

    nodes = list(graph.nodes())
visited = {node: False for node in nodes}
    discovery_time = {node: float('inf') for
                                node in nodes}
low = {node: float('inf') for node in nodes}
parent = {node: None for node in nodes}
ap = {node: False for node in nodes}
                time = [0]

            for node in nodes:
            if not visited[node]:
```

```
                dfs(node, parent, discovery_time,
                    low, ap, visited, time, 0)

        articulation_points = [node for node,
                is_ap in ap.items() if is_ap]
        return articulation_points

    # Main function to check if graph is 2-
            vertex strongly biconnected
    def
is_2_vertex_strongly_biconnected(graph):
        # Check if the graph is strongly
                connected
        sccs = gabow_scc(graph)
        if len(sccs) > 1:
            return False

    # Convert directed graph to undirected
            graph
    undirected_graph = graph.to_undirected()
```

```
    # Check for articulation points in the
undirected graph
    articulation_points =
jens_schmidt_ap(undirected_graph)
    if articulation_points:
        return False

    return True

# Example usage
if __name__ == "__main__":
    # Example graph
    G = nx.DiGraph()
    G.add_edges_from([(1, 2), (2, 3), (3, 1),
(2, 4), (4, 5), (5, 2)])

# Check if the graph is 2-vertex strongly
biconnected
result =
is_2_vertex_strongly_biconnected(G)
print("The graph is 2-vertex strongly
```

```
biconnected:", result)

# Plot the graph for visual verification
nx.draw(G, with_labels=True,
node_color='lightblue', edge_color='gray')
plt.show()
```