# assignment-3-1

February 12, 2024

Problem

You are given an integer n. Determine if there is an unconnected graph with n vertices that contains at least two connected components and contains the number of edges that is equal to the number of vertices. Each vertex must follow one of these conditions:

- Its degree is less than or equal to 1.
- It's a cut-vertexLinks to an external site..

Input format:

- First line: n

Output format

- Print Yes if it is an unconnected graph. Otherwise, print No.

graph_data = { 1: {3}, 2: {3}, 3:{1, 2, 4}, 4:{3, 5, 6}, 5:{4, 6, 7}, 6:{4, 5, 8}, 7:{5}, 8:{6} } Cut vertex is 3, 4, 5, and 6 Component: 1

[14]:
```python
class Graph:

    def __init__(self, data=None):
        self.graph_data = data or {}

    def all_edges(self, vertex):
        return self.graph_data.get(vertex, set())

    def all_vertices(self, vertex):
        self.graph_data.setdefault(vertex, set())

    def add_edge(self, edge):
        edge = set(edge)
        vertex1, vertex2 = tuple(edge)
        for x, y in [(vertex1, vertex2), (vertex2, vertex1)]:
            if x in self.graph_data:
                self.graph_data[x].add(y)
            else:
                self.graph_data[x] = {y}
```

```
    def dfs(self, v, visited, component):
        visited.add(v)
        component.add(v)
        for neighbor in self.all_edges(v):
            if neighbor not in visited:
                self.dfs(neighbor, visited, component)

    def graphing(self):
        visited = set()
        components = []

        for v in self.graph_data:
            if v not in visited:
                component = set()
                self.dfs(v, visited, component)
                components.append(component)

        print("============================================================")
        print("Graph Components:")
        for i, components in enumerate(components, start=1):
            print(f"component {i}: {components}")

    def find_path(self, start_vertex, end_vertex, path=None):
        if path is None:
            path = []
        graph = self.graph_data
        path = path +[start_vertex]
        if start_vertex not in graph:
            return None
        if start_vertex == end_vertex:
            return path
        for vertex in graph[start_vertex]:
            if vertex not in path:
                extended_path = self.find_path(vertex, end_vertex, path)
                if extended_path:
                    return extended_path
        return None

    def check_graph(self):
        list_path = []
        for vertex in self.graph_data:
            for next_vertex in self.graph_data:
                if vertex != next_vertex:
                    path = self.find_path(vertex, next_vertex)
                    if path is None:
                        list_path.append(path)
                    elif path is not None:
```

```
                        list_path.extend(path)
        if None in list_path:
            print("YES")
        elif None not in list_path:
            print("NO")

    def connections(self):
        print("============================================================")
        print("Connected Vertices:")
        for vertex in self.graph_data:
            connections = self.graph_data[vertex]
            print(f"Vertex {vertex} is connected to {connections if connections␣
 ↪else 'no other vertices'}")
        print("============================================================")
        print("Is the graph Unconnected?")
        self.check_graph()


    def edges(self):
        edges_list = []
        for vertex in self.graph_data:
            for neighbour in self.graph_data[vertex]:
                if {neighbour, vertex} not in edges_list:
                    edges_list.append({vertex, neighbour})
        return edges_list

if __name__ == "__main__":
  graph_data = {
      1: {3},
      2: {3},
      3: {1, 2, 4},
      4: {3, 5, 6},
      5: {4, 6, 7},
      6: {4, 5, 8},
      7: {5},
      8: {6}
  }
  print("============================================================")
  print("The graph: \n", graph_data)
  adornado = Graph(graph_data)
  adornado.connections()
  adornado.graphing()
  print("============================================================")
  print("Graph Edges:")
  print(adornado.edges())
  print("============================================================")
```

```
=========================================================
The graph:
 {1: {3}, 2: {3}, 3: {1, 2, 4}, 4: {3, 5, 6}, 5: {4, 6, 7}, 6: {8, 4, 5}, 7:
{5}, 8: {6}}
=========================================================
Connected Vertices:
Vertex 1 is connected to {3}
Vertex 2 is connected to {3}
Vertex 3 is connected to {1, 2, 4}
Vertex 4 is connected to {3, 5, 6}
Vertex 5 is connected to {4, 6, 7}
Vertex 6 is connected to {8, 4, 5}
Vertex 7 is connected to {5}
Vertex 8 is connected to {6}
=========================================================
Is the graph Unconnected?
NO
=========================================================
Graph Components:
component 1: {1, 2, 3, 4, 5, 6, 7, 8}
=========================================================
Graph Edges:
[{1, 3}, {2, 3}, {3, 4}, {4, 5}, {4, 6}, {5, 6}, {5, 7}, {8, 6}]
=========================================================
```

In conclusion the build a graph assignment is quite hard or really hard its a good thing that some of the codes are already given to us which i made use of like the path fin, checker, the edges and i also did a research about this code where they use dfs which i also implemented in my code, because its a algorithm that is also good for using in graphs like this. I also use a drawing to analyze how my graph would be and i also planned it first before doing the code which is hard takes a lot of practices to learn and do the codes that are given to us. In my code I implemented the displaying of the graph, showing the connected vertices, graph checker, graph components where it will display if the graph is unconnected the graph components will show how many components there is and if the graph is connected the it will show 1 component, and also we have graph edges where we have a graph containing 8 vertex and edges should also have the same amount of edges which are the lines so we have the graph edges.