Assignment 3.1 Practice Problem 1 (Build a Graph)

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:

1. Its degree is less than or equal to 1.
2. It's a cut-vertex.

Note:

- The graph must be simple.
- Loops and multiple edges are not allowed.

Input Format

- First line: n.

Output Format

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

Constraints

- 1 <= n <= 100

In [83]:

```
class Graph(object):

    def __init__(self, graph_dict=None):
        if graph_dict == None:
            graph_dict = {}
        self._graph_dict = graph_dict

    def edges(self, vertice):
        return self._graph_dict[vertice]

    def all_vertices(self):
        return set(self._graph_dict.keys())

    def all_edges(self):
        return self.__generate_edges()

    def add_vertex(self, vertex):
        if vertex not in self._graph_dict:
            self._graph_dict[vertex] = []

    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_dict:
                self._graph_dict[x].add(y)
            else:
                self._graph_dict[x] = [y]

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

    def __iter__(self):
        self._iter_obj = iter(self._graph_dict)
        return self._iter_obj

    def __next__(self):
        """ allows us to iterate over the vertices """
        return next(self._iter_obj)

    def __str__(self):
        res = "vertices: "
        for k in self._graph_dict:
            res += str(k) + " "
        res += "\nedges: "
        for edge in self.__generate_edges():
            res += str(edge) + " "
        return res

# 1 <= n <= 100 (constraint)
def unconnected_checker(graph):
    if len(graph.all_vertices()) == 1:
        return "No"  # no multiple connected components

    visited = set()
    connected_components = 0

    def dfs(vertex):
      visited.add(vertex)
      for neighbour in graph.edges(vertex):
        if neighbour not in visited:
          dfs(neighbour)

    for vertex in graph.all_vertices():
        if vertex not in visited:
            dfs(vertex)
            connected_components += 1

    if connected_components >= 2:
        return "Yes"
    else:
        return "No"
```

In [84]:

```
"""

a ----- b
```

```
|       |
|       |
c ----- d

"""
g = {
    'a': {'b', 'c'},
    'b': {'a', 'd'},
    'c': {'a', 'd'},
    'd': {'b', 'c'}
}

"""


a ---- b    e ---- f
|      |    |      |
|      |    |      |
c      d    g      h

"""
g2 = {
    'a': {'b', 'c'},
    'b': {'a', 'd'},
    'c': {'a'},
    'd': {'b'},
    'e': {'f', 'g'},
    'f': {'e', 'h'},
    'g': {'e'},
    'h': {'f'}
}

g3 = {
    'a': {'b', 'c'},
    'b': {'a'},
    'c': {'a'},
    'd': { },
    'e': {'f'},
    'f': {'e'},
    'g': { },
    'h': { }
}

g4 = {
    'a': {'b'},  # 'a' = degree 1
    'b': {'a'},  # 'b' = degree 1
    'c': {},     # 'c' = degree 0
}
```

In [90]:

```
graph = Graph(g)
result = unconnected_checker(graph)


for vertice in graph:
  print(f"Edges of vertice {vertice}: ", graph.edges(vertice))

print("\nVertices of graph:")
print(graph.all_vertices())

print("\nEdges of graph:")
print(graph.all_edges())

print("\nUnconnected Graph Checker:")
print(result)
```

```
Edges of vertice a:  {'b', 'c'}
Edges of vertice b:  {'d', 'a'}
Edges of vertice c:  {'d', 'a'}
Edges of vertice d:  {'b', 'c'}

Vertices of graph:
{'d', 'b', 'a', 'c'}

Edges of graph:
[{'b', 'a'}, {'a', 'c'}, {'d', 'b'}, {'d', 'c'}]

Unconnected Graph Checker:
No
```

In [91]:

```
graph = Graph(g2)
result = unconnected_checker(graph)


for vertice in graph:
  print(f"Edges of vertice {vertice}: ", graph.edges(vertice))

print("\nVertices of graph:")
print(graph.all_vertices())

print("\nEdges of graph:")
print(graph.all_edges())

print("\nUnconnected Graph Checker:")
print(result)
```

```
Edges of vertice a:  {'b', 'c'}
Edges of vertice b:  {'d', 'a'}
Edges of vertice c:  {'a'}
Edges of vertice d:  {'b'}
Edges of vertice e:  {'f', 'g'}
Edges of vertice f:  {'h', 'e'}
Edges of vertice g:  {'e'}
Edges of vertice h:  {'f'}

Vertices of graph:
{'f', 'h', 'd', 'c', 'e', 'g', 'b', 'a'}

Edges of graph:
[{'b', 'a'}, {'a', 'c'}, {'d', 'b'}, {'f', 'e'}, {'g', 'e'}, {'h', 'f'}]

Unconnected Graph Checker:
Yes
```

In [92]:

```
graph = Graph(g3)
result = unconnected_checker(graph)


for vertice in graph:
  print(f"Edges of vertice {vertice}: ", graph.edges(vertice))

print("\nVertices of graph:")
print(graph.all_vertices())

print("\nEdges of graph:")
print(graph.all_edges())

print("\nUnconnected Graph Checker:")
print(result)
```

```
Edges of vertice a:  {'b', 'c'}
Edges of vertice b:  {'a'}
Edges of vertice c:  {'a'}
Edges of vertice d:  {}
Edges of vertice e:  {'f'}
Edges of vertice f:  {'e'}
Edges of vertice g:  {}
Edges of vertice h:  {}

Vertices of graph:
{'f', 'h', 'd', 'c', 'e', 'g', 'b', 'a'}

Edges of graph:
[{'b', 'a'}, {'a', 'c'}, {'f', 'e'}]

Unconnected Graph Checker:
Yes
```

In [93]:

```
graph = Graph(g4)
result = unconnected_checker(graph)


for vertice in graph:
  print(f"Edges of vertice {vertice}: ", graph.edges(vertice))

print("\nVertices of graph:")
print(graph.all_vertices())

print("\nEdges of graph:")
print(graph.all_edges())

print("\nUnconnected Graph Checker:")
print(result)
```

```
Edges of vertice a:  {'b'}
Edges of vertice b:  {'a'}
Edges of vertice c:  {}

Vertices of graph:
{'b', 'a', 'c'}

Edges of graph:
[{'b', 'a'}]

Unconnected Graph Checker:
Yes
```