**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:

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

It's a cut-vertexLinks to an external site...

**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

```
### graph checker ###
class Graph:
    def __init__(self, grp):
        self.grp = grp

    def is_unc(self): # if graph is unconnected
        visited = set() #tracks visited vertices
        components = 0 #counter fo graph nodes

        def dfs(vx): #Depth first search
            visited.add(vx) #visited nodes marker
            for ni in self.grp[vx]: # iterates through neighbors of the current node
              if ni not in visited: # explores  unvisisted nodes
                  dfs(ni)

        for vx in self.grp: # iterates through all the nodes/vertices in the graph
          if vx not in visited: # explores unvisisted nodes/ apply dfs upon it
            dfs(vx)
            components += 1 # increment component counter when exploration is done

        return components > 1 # if there components > 1 then it is not connected
### graph checker ###


### graph builder ###
def create_gdict():
  n = int(input("Enter number of vertices: "))
  if n >= 100:
      print("Vertices over 100 is not allowed!")
  else:
    gdict = {} # dictionary initiation

    for i in range (1, n + 1): #iteration fo the vertices input
      vertex = input(f"Enter Vertex {i}: ")
      neighbors = input(f"Enter the neighbors of vertex {i}: ").split()
      gdict[vertex] = set(neighbors)

    return gdict #returns the graph dictionary

gdict = create_gdict() #This would create the graph dictionary
### graph builder ###

### class implementation ###
gp = Graph(gdict) # converts the user input dictionary into a graph object

if gp.is_unc():
  print("Yes\n") #if graph is not connected
else:
  print("No\n") #if graph is connected
```

```
Enter number of vertices: 6
Enter Vertex 1: 1
Enter the neighbors of vertex 1: 2
```

```
Enter Vertex 2: 2
Enter the neighbors of vertex 2: 1
Enter Vertex 3: 3
Enter the neighbors of vertex 3: 4
Enter Vertex 4: 4
Enter the neighbors of vertex 4: 3
Enter Vertex 5: 5
Enter the neighbors of vertex 5: 6
Enter Vertex 6: 6
Enter the neighbors of vertex 6: 5
Yes
```

```
### graph checker ###
class Graph:
    def __init__(self, grp):
        self.grp = grp

    def is_unc(self): # if graph is unconnected
        visited = set() #tracks visited vertices
        components = 0 #counter fo graph nodes

        def dfs(vx): #Depth first search
            visited.add(vx) #visited nodes marker
            for ni in self.grp[vx]: # iterates through neighbors of the current node
              if ni not in visited: # explores  unvisisted nodes
                  dfs(ni)

        for vx in self.grp: # iterates through all the nodes/vertices in the graph
          if vx not in visited: # explores unvisisted nodes/ apply dfs upon it
            dfs(vx)
            components += 1 # increment component counter when exploration is done

        return components > 1 # if there components > 1 then it is not connected
### graph checker ###


### graph builder ###
def create_gdict():
  n = int(input("Enter number of vertices: "))
  if n >= 100:
      print("Vertices over 100 is not allowed!")
  else:
    gdict = {} # dictionary initiation

    for i in range (1, n + 1): #iteration fo the vertices input
      vertex = input(f"Enter Vertex {i}: ")
      neighbors = input(f"Enter the neighbors of vertex {i}: ").split() # note seperate the vertices via spaces!!!
      gdict[vertex] = set(neighbors)

    return gdict #returns the graph dictionary

gdict = create_gdict() #This would create the graph dictionary
### graph builder ###

### class implementation ###
gp = Graph(gdict) # converts the user input dictionary into a graph object

if gp.is_unc():
  print("Yes\n") #if graph is not connected
else:
  print("No\n") #if graph is connected
```

```
Enter number of vertices: 6
Enter Vertex 1: 1
Enter the neighbors of vertex 1: 2 3 4
Enter Vertex 2: 2
Enter the neighbors of vertex 2: 1 3 5
Enter Vertex 3: 3
Enter the neighbors of vertex 3: 1 2 6
Enter Vertex 4: 4
Enter the neighbors of vertex 4: 1
Enter Vertex 5: 5
Enter the neighbors of vertex 5: 2
Enter Vertex 6: 6
Enter the neighbors of vertex 6: 3
No
```