Project 2 – Cycle Detection

In this project, you are asked to implement functions that will detect and determine cycles from directed or undirected graph.

The objective of this project is to learn about basics of graphs, depth first search trees, topological sort, and get familiar with graphs and the cycle detection problem.

Project Design Rules:

- You can use any modules that are default in python, which means a module that is already available in python and do not need any installation. Submissions that use any other module or file won't be graded.
- You will be graded for each of the given functions so make sure to implement each of the functions **with exact names, parameters, and default values** -if given- in your python code.
- **Graphs:** The functions require a graph as an input. Graphs are defined as an `object` that has following properties:
  - **`self.add_edge(u,v)`:** takes vertices of an edge `u` and `v`, and add them to a `dictionary` that holds adjacency relation as key and value pairs.
  - **`self. vertices`:** a `list` that holds the vertices used in the graph.
  - **`self.adj[u]`:** a `dictionary` that holds the neighbors of vertex `u`.
  - **`self.gtype`:** a `string` that identifies the type of the graph; it can be either `'u'` for undirected graphs or `'d'` for directed graphs.

  Note that all functions will be tested with the graph object defined above and any functions that do not conform with the graph definitions given above, will be graded with '0' points.

Functions:

Function Name	**Explanation**
`Dir_cyclic_or_acyclic(Graph)`	This function takes a graph object as an input, check the graph as it is directed or not, then determines if the graph is cyclic or not. If graph has cycle or cycles it returns `True`, if not returns `False`. If the graph is not a directed graph, returns `None`. Input: • `Graph : object` Return a `boolean` or `None`.

`UnDir_cyclic_or_acyclic(Graph)`	This function takes a graph object as an input, check the graph as it is undirected or not, then determines if the graph is cyclic or not. If graph has cycle or cycles it returns `True`, if not returns `False`. If the graph is not an undirected graph, returns `None`. Input: ● `Graph` : object Return a `boolean` or `None`.
`T_sort(Graph)`	This function takes a graph object as an input, check the graphs is a directed acyclic graph (DAG) or not. If the graph is a DAG, then returns a list that hold the topological sort of vertices. If the graph is not a DAG, returns `None`. Input: ● `Graph` : object Return a list or `None`.

```
>>> print(Gu1.gtype)
u
>>> print(Gu1.vertices)
[4, 1, 2, 3, 5, 6]
>>> print(Gu1.adj)
{4: [1, 5], 1: [4, 2], 2: [1, 3, 5], 3: [2, 6], 5: [2, 6, 4], 6: [3, 5]}
>>> print(UnDir_cyclic_or_acyclic(Gu1))
True
>>> print(Gu2.gtype)
u
>>> print(Gu2.vertices)
[1, 0, 2, 3, 4]
>>> print(Gu2.adj)
{1: [0, 2], 0: [1, 3], 2: [1], 3: [0, 4], 4: [3]}
>>> print(UnDir_cyclic_or_acyclic(Gu2))
False
>>> print(Gu3.gtype)
u
>>> print(Gu3.vertices)
[1, 0, 2, 3, 4]
>>> print(Gu3.adj)
{1: [0, 2], 0: [1, 2, 3], 2: [1, 0], 3: [0, 4], 4: [3]}
>>> print(UnDir_cyclic_or_acyclic(Gu3))
True
>>> print(Gd1.gtype)
d
```

```
>>> print(Gd1.vertices)
[4, 1, 2, 3, 5, 6]
>>> print(Gd1.adj)
{4: [1, 5], 1: None, 2: [1, 3, 5], 3: [6], 5: None, 6: [5]}
>>> print(UnDir_cyclic_or_acyclic(Gd1))
None
>>> print(Dir_cyclic_or_acyclic(Gd1))
True
>>> print(Gd2.gtype)
d
>>> print(Gd2.vertices)
[0, 1, 2, 3]
>>> print(Gd2.adj)
{0: [1, 2], 1: [2], 2: [3], 3: None}
>>> print(Dir_cyclic_or_acyclic(Gd2))
False
>>> print(Gd3.gtype)
d
>>> print(Gd3.vertices)
[0, 1, 2, 3]
>>> print(Gd3.adj)
{0: [1, 2], 1: [2], 2: [0, 3], 3: None}
>>> print(Dir_cyclic_or_acyclic(Gd3))
True
>>> print(Gd4.gtype)
d
>>> print(Gd4.vertices)
[0, 1, 2, 3]
>>> print(Gd4.adj)
{0: [1, 2], 1: [2], 2: [3], 3: [3]}
>>> print(Dir_cyclic_or_acyclic(Gd4))
True
>>> print("T_sort examples")
T_sort examples
>>> print(T_sort(Gd1))
None
>>> print(T_sort(Gu1))
None
>>> print(T_sort(Gd2))
[0, 1, 2, 3]
```