Assignment 10.1 Sub-Graphs (2 points)

Your task is to implement a graph structure in Python. The graph can be modified by creating or removing edges between vertices. If two vertices can be reached by edges, they belong to the same connected component. The graph should be able determine how many connected components it contains.

Create a class `Graph` which has all the following methods:

- `__init__(n)`: initialize a new `Graph` object with `n` vertices
- `add(u, v)`: creates an undirected edge between vertex `u` and `v`
- `remove(u, v)`: removes an edge between vertex `u` and `v`
- `subgraphs()`: returns the number of connected components in the graph

Vertices in the graph are labeled from 0 to $n-1$ as integers (`int`).

A code template with an example program:

```
# subgraphs.py


class Graph:
    # TODO


if __name__ == "__main__":
graph = Graph(6)

    edges = ((0, 4), (2, 1), (2, 5), (3, 0), (5, 1))
    for u, v in edges:
        graph.add(u, v)

    print(graph.subgraphs())

    more_edges = ((0, 2), (2, 3), (3, 5), (4, 5))
    for u, v in more_edges:
        graph.add(u, v)

    print(graph.subgraphs())
```

```
$ python subgraphs.py
2
1
```

Submit your solution in CodeGrade as `subgraphs.py`.

Assignment 10.2: All Paths (3 points)

Your task is to implement a graph structure in Python. The graph can be modified by creating or removing edge of certain weight between two vertices. Additionally, the graph can calculate and output $n \times n$ matrix of shortest paths (least weight) between all vertices.

Create a class `Graph` which has all the following methods:

- `__init__(n)`: initialize a new `Graph` object with `n` vertices
- `add(u, v, w)`: creates a directed edge from vertex `u` to `v` with weight `w`
- `remove(u, v)`: removes an edge from vertex `u` to `v`
- `all_paths()`: returns a $n \times n$ `list` of all paths with least weight between each vertex, nonexistent paths are given value of `-1`

Vertices in the graph are labeled from 0 to $n - 1$ as integers (`int`).

A code template with an example program:

```
# paths.py


class Graph:
    # TODO


if __name__ == "__main__":
    graph = Graph(6)
    edges = ((0, 2, 7), (0, 4, 9), (2, 1, 5),
                    (2, 3, 1), (2, 5, 2), (3, 0, 6),
                    (3, 5, 2), (4, 5, 1), (5, 1, 6))
    for u, v, w in edges:
        graph.add(u, v, w)

    M = graph.all_paths()
    for weights in M:
        for weight in weights:
            print(f"{weight:3d}", end="")
        print()
```

Output:

```
$ python paths.py
 0 12  7  8  9  9
-1  0 -1 -1 -1 -1
 7  5  0  1 16  2
 6  8 13  0 15  2
-1  7 -1 -1  0  1
-1  6 -1 -1 -1  0
```

Submit your solution in CodeGrade as `paths.py`.

Assignment 10.3 Minimal Cost Spanning Tree (3 points)

Your task is to implement a graph structure in Python. On any time, the graph can be modified by creating or removing edge of certain weight between two vertices. Additionally, it is possible to determine what is the minimal cost spanning tree and to calculate its total weight.

Create a class `Graph` which has all the following methods:

- `__init__(n)`: initialize a new `Graph` object with `n` vertices
- `add(u, v, w)`: creates an undirected edge between vertex `u` and `v` with weight `w`
- `remove(u, v)`: removes an edge between vertex `u` and `v`
- `min_cost(start, end)`: computes a total cost of minimal weight spanning tree of the graph.

A code template with an example program:

```
# mincost.py


class Graph:
    # TODO


if __name__ == "__main__":
    graph = Graph(6)
    edges = ((0, 2, 7), (0, 4, 9), (2, 1, 5),
                    (2, 3, 1), (2, 5, 2), (3, 0, 6),
                    (3, 5, 2), (4, 5, 1), (5, 1, 6))
    for u, v, w in edges:
        graph.add(u, v, w)

    print(graph.min_cost())

    graph.remove(2, 3)

    print(graph.min_cost())
```

Output:

```
$ python mincost.py
15
16
```

Submit your solution in CodeGrade as `mincost.py`.

Last modified: Monday, 18 November 2024, 11:44 AM