Algorithms: Design and Analysis - CS 412

Weekly Challenge 07: Maximum Flows

Ali Muhammad Asad - aa07190

We are going to write a class `MaxFlow` in a file `maxflow.py`. Objects of the class will take in a flow network and compute its maximum flow. Your code will be tested by `pytest` using the file, `test_maxflow.py`, given in `WC7_MaxFlows.zip`. To test your implementation of `MaxFlow`, open the directory containing `test_maxflow.py` and `maxflow.py` in the terminal, and run the following command:

```
pytest test_maxflow.py
```

TASKS:

(a) The `__init__` method of `MaxFlow` takes as argument a `file` object which points to a file containing a flow network, G, in the following format.

- The first line contains the number of edges, e.
- Each of the next e lines contains an edge in the format u v c where (u, v) is an edge in G with capacity c.
- The source vertex is always named, s, and the sink vertex is always named, t.
- All capacities are integral and positive.

(b) The `get_value` method returns the value of the maximum flow on G.

(c) The `get_flow` method returns the maximum flow on G as a `dict` object. A key in the returned `dict` object is an edge, (u, v) in the maximum flow and the value is the amount of flow along (u, v). Only edges with non-zero flow are included.

(d) Ensure that all tests pass by running `pytest` locally.

(e) Do not include any external packages except `networkx` for the possible use of `networkx.DiGraph` to store and operate on the flow network.

(f) You may modify the error messages in `test_maxflow.py` to convey more information if you wish, but you may not alter any other functionality in it.

Solution:

```
import networkx as nx
class MaxFlow:
    def __init__(self, file) -> None:
        self.dag = nx.DiGraph()
        for i in range(len(file)):
            node1, node2, capacity = file[i].split()
            self.dag.add_edge(node1, node2, capacity=int(capacity))
            if not self.dag.has_edge(node2, node1):
                self.dag.add_edge(node2, node1, capacity=0)

    def get_value(self):
        ''' Get the maximum flow value of the DAG '''
        residual = self.dag.copy()
        parent = {}

        max_flow = 0

        while self.bfs(residual, 's', 't', parent):
            path_flow = float("Inf")
            s = 't'
            while s != 's':
                path_flow = min(path_flow, residual[parent[s]][s]['capacity
    '])
                s = parent[s]
            max_flow += path_flow
            v = 't'
            while v != 's':
                u = parent[v]
                residual[u][v]['capacity'] -= path_flow
                residual[v][u]['capacity'] += path_flow
                v = parent[v]
        return max_flow

    def bfs(self, residual, source, sink, parent):
        visited = {node: False for node in residual.nodes()}
        queue = []
        queue.append(source)
        visited[source] = True

        while queue:
            u = queue.pop(0)
            for ind, val in enumerate(residual[u]):
                if visited[val] == False and residual[u][val]['capacity'] >
    0:
                    queue.append(val)
                    visited[val] = True
                    parent[val] = u

        return True if visited[sink] else False

    def get_flow(self):
        ''' Get the flow of each node in the DAG '''
        residual = self.dag.copy()
        parent = {}

```

```
        flow = {}
        for u, v in self.dag.edges():
            flow[(u, v)] = 0

        while self.bfs(residual, 's', 't', parent):
            path_flow = float("Inf")
            s = 't'
            path = []
            while s != 's':
                path.append((parent[s], s))
                path_flow = min(path_flow, residual[parent[s]][s]['capacity
    '])
                s = parent[s]

            for u, v in path:
                flow[(u, v)] += path_flow

            v = 't'
            while v != 's':
                u = parent[v]
                residual[u][v]['capacity'] -= path_flow
                residual[v][u]['capacity'] += path_flow
                v = parent[v]

        flow = {edge: f for edge, f in flow.items() if f > 0}
        return flow

```